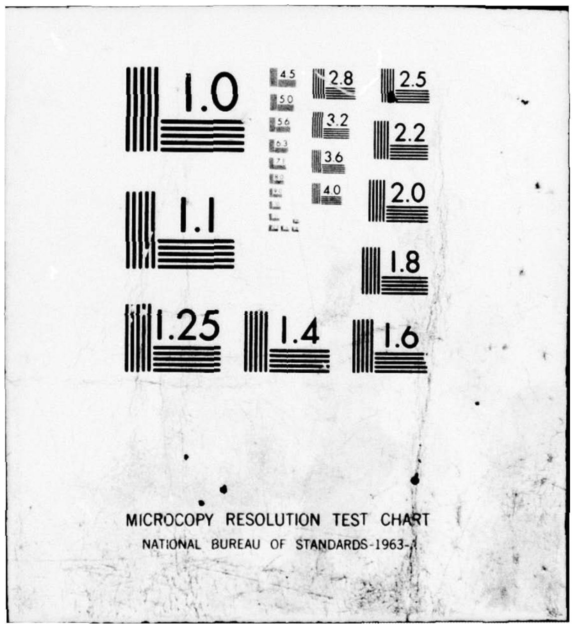
AD-A078 690 PURDUE UNIV LAFAYETTE IN SCHOOL OF ELECTRICAL ENGINEERING F/G 9/2
DESIGN AND ANALYSIS OF INTERCONNECTION NETWORKS FOR PARTITIONAB--ETC(U)
AUG 79 S D SMITH , H J SIEGEL AFOSR-78-3581
UNCLASSIFIED TR-EE79-39 AFOSR-TR-79-1272 NL

1 OF 4
AD
A078690



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

4

DESIGN AND ANALYSIS OF INTERCONNECTION NETWORKS FOR PARTITIONABLE PARALLEL PROCESSING SYSTEMS

S. Diane Smith

Howard Jay Siegel

School of Electrical Engineering
Purdue University
West Lafayette, Indiana

TR-EE 79-39
August 1979

This work was supported in part by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant number AFOSR-78-3581 and by the Purdue Research Foundation under a David Ross Grant (PRF #8718).

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) Unclassified

REPORT DOCUMENTATION PAGE 18		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 19 AFOSR-TR-79-1272	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 9
4. TITLE (and Subtitle) 6 DESIGN AND ANALYSIS OF INTERCONNECTION NETWORKS FOR PARTITIONABLE PARALLEL PROCESSING SYSTEMS.		5. TYPE OF REPORT & PERIOD COVERED Interim rept.
		6. PERFORMING ORG. REPORT NUMBER 14 TR-EE 79-39
7. AUTHOR(s) 10 S. Diane Smith, Howard Jay Siegel		8. CONTRACT OR GRANT NUMBER(s) 15 AFOSR-78-3581
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Electrical Engineering, Purdue University, West Lafayette, IN 47907		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 16 2304 17 A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM, Bolling AFB, Washington, DC 20332		12. REPORT DATE 11 August 1979
		13. NUMBER OF PAGES 293
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 297		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computer architecture, cube network, dynamically reconfigurable systems, image processing, interconnection networks, MIMD machine, multimicroprocessor systems, parallel processing, partitionable systems, PM2I network, shuffle-exchange, SIMD machine.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Please see reverse side.

DD FORM 1 JAN 73 1473

292000 UNCLASSIFIED

LB

20. ABSTRACT

A single instruction stream - multiple data stream (SIMD) computer performs one algorithm (single instruction stream) on vectors of data (multiple data stream). The model of an SIMD machine used here consists of a control unit (CU), processing elements (PEs), and an interconnection network. The CU broadcasts instructions to the N PEs (where N is a power of two). The interconnection network is the mechanism that allows PEs to pass data among themselves.

Four types of interconnection networks are discussed in this work: the Shuffle-Exchange network, the Cube network, the ILLIAC network, and the Plus-Minus 2^i (PM2I) network. Each type has been discussed in the literature and used in an existing or proposed machine design.

For each of these four network types, different hardware structures are considered. A recirculating network consists of one stage of switches that is reused until the data reach their final destinations. A combinational logic multistage network consists of several stages of switches and, usually, data is transferred in one pass through the network. In pipelined multistage networks, which are introduced, registers are inserted after each stage of a combinational logic multistage network. The data are divided into segments, and these segments are passed in a parallel-pipelined manner. Hardware implementations for recirculating, combinational logic multistage, and pipelined multistage networks are presented and analyzed.

Some problems may be more efficiently solved if a large SIMD machine can be partitioned into smaller groups of varying sizes of powers of two. The interconnection network must be able to support this partitioned machine. The partitioning properties of the four types of networks are presented.

In the selection of an interconnection network for a computer design, the types of algorithms that must be executed should be considered. A detailed analysis of various networks is presented for three parallel image processing algorithms: smoothing, histogram formation, and data classification.

The Augmented Data Manipulator (ADM) network is introduced. An analysis is presented which compares the capabilities of the ADM with those of a multistage Cube network and with the Inverse Augmented Data Manipulator network.

The Emulator System, a proposed hardware design aid, is introduced. The flexibility and power of this system is demonstrated by its ability to simulate many types of interconnection networks and control schemes that have appeared in the literature.

As the costs of microprocessors continue to decrease, more large scale multiprocessor systems are being proposed and built. This thesis will aid system architects in designing a partitionable interconnection network appropriate for their particular needs.

4

DESIGN AND ANALYSIS OF INTERCONNECTION NETWORKS FOR PARTITIONABLE PARALLEL PROCESSING SYSTEMS

S. Diane Smith

Howard Jay Siegel

Purdue University

School of Electrical Engineering

West Lafayette, IN 47907

Purdue University

TR-EE 79-39

August 1979

This work was supported in part by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant number AFOSR-78-3581 and by the Purdue Research Foundation under a David Ross Grant (PRF#8718).

ACKNOWLEDGEMENTS

The author wishes to thank the following people for their comments on the content of this work: Prof. K. S. Fu, Prof. A. P. Reeves, Prof. C. Smith, Prof. L. J. Siegel, R. J. McMillen, and P. T. Mueller, Jr. This work was supported in part by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant number AFOSR-78-3581 and by the Purdue Research Foundation under a David Ross Grant (PRF#8718).

Portions of this work have appeared in the following papers:

H. J. Siegel and S. Diane Smith, "Study of Multistage SIMD Interconnection Networks," _Proceedings of the Fifth Annual Symposium on Computer Architecture_, pp. 223-229, April 1978.

S. Diane Smith and H. J. Siegel, "Recirculating, Pipelined, and Multistage SIMD Interconnection Networks," _Proceedings of the 1978 International Conference on Parallel Processing_, pp. 206-214, August 1978.

S. Diane Smith and H. J. Siegel, "An Emulator Network for SIMD Machine Interconnection Networks," _Proceedings of the 1979 International Symposium on Computer Architecture_, pp. 232-241, April 1979.

H. J. Siegel, Leah J. Siegel, Robert J. McMillen, Philip T. Mueller, Jr., and S. Diane Smith, "An SIMD/MIMD Multimicroprocessor System for Image Processing and Pattern Recognition," _Proceedings of the 1979 IEEE Computer Society Conference on Pattern Recognition and Image Processing_, pp. 214-224, August 1979.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

PAGE

LIST OF TABLES

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

A single instruction stream - multiple data stream (SIMD) computer performs one algorithm (single instruction stream) on vectors of data (multiple data stream). The model of an SIMD machine used here consists of a control unit (CU), processing elements (PEs), and an interconnection network. The CU broadcasts instructions to the N PEs (where N is a power of two). The interconnection network is the mechanism that allows PEs to pass data among themselves.

Four types of interconnection networks are discussed in this work: the Shuffle-Exchange network, the Cube network, the ILLIAC network, and the Plus-Minus 2^i (PM2I) network. Each type has been discussed in the literature and used in an existing or proposed machine design.

For each of these four network types, different hardware structures are considered. A recirculating network consists of one stage of switches that is reused until the data reach their final destinations. A combinational logic multistage network consists of several stages of switches and, usually, data is transferred in one pass through the network. In pipelined multistage networks, which are introduced, registers are inserted after each stage of a combinational logic multistage network. The data are divided into segments, and these segments are passed in a parallel-pipelined manner. Hardware implementations for recirculating, combinational logic multistage, and pipelined multistage networks are presented and analyzed.

Some problems may be more efficiently solved if a large SIMD machine can be partitioned into smaller groups of varying sizes of powers of two. The interconnection network must be able to support this partitioned machine. The partitioning properties of the four types of networks are presented.

In the selection of an interconnection network for a computer design, the types of algorithms that must be executed should be considered. A detailed analysis of various networks is presented for three parallel image processing algorithms: smoothing, histogram formation, and data classification.

The Augmented Data Manipulator (ADM) network is introduced. An analysis is presented which compares the capabilities of the ADM with those of a multistage Cube network and with the Inverse Augmented Data Manipulator network.

The Emulator System, a proposed hardware design aid, is introduced. The flexibility and power of this system is demonstrated by its ability to simulate many types of interconnection networks and control schemes that have appeared in the literature.

As the costs of microprocessors continue to decrease, more large scale multiprocessor systems are being proposed and built. This thesis will aid system architects in designing a partitionable interconnection network appropriate for their particular needs.

I. INTRODUCTION

Two basic factors influence the speed of operation of a computer system. First is the speed of the logic circuits. Future technology promises to bring this into the picosecond delay range, but theoretical limitations, such as the speed of light, mean that other methods should be used to increase computational speed. The second factor, then, is the organization of the machine and the algorithms which it performs.

Inexpensive microprocessors have made large scale parallel processing systems feasible. Such architectures can be used for problems that can be broken into independent subtasks, which can be done simultaneously, thus increasing computational speed. Examples of problems that benefit from parallel processing systems are weather forecasting, image processing, and air traffic control.

One type of parallel architecture is an SIMD (single instruction stream - multiple data stream) system. The main components of this type of system are

(1) N processing elements which operate in parallel, all executing the same instruction at the same time,

(2) one control unit (CU), which sends instructions and other control information to the processing elements, and

(3) an interconnection network, which allows the processing elements to communicate among themselves.

Many questions remain unanswered about the design and use of interconnection networks. This research formulates design criteria and analysis techniques for interconnection networks with emphasis placed on a class of multiprocessor systems which we call partitionable parallel processing systems. A partitionable parallel processing system is a reconfigurable parallel computer which can be configured not only as one SIMD machine with N processing elements, but also as many smaller SIMD machines. If T tasks each require at most N/T processing elements, then this multiple SIMD mode more fully utilizes the system.

Chapter II introduces the terminology of parallel systems and interconnection networks which will be used throughout this research. A survey of the background literature in the field of interconnection networks is presented in Chapter III. Some of the networks which are discussed and analyzed in this work are introduced in Chapter III.

Chapter IV considers various aspects of the structure of networks and the circuits used to build them. Single stage networks and multistage networks of combinational logic are designed. By inserting registers after stages of a multistage network, blocks of data can be pipelined through the network to improve the effective throughput of the data transfer. The effects of pipelining on the cost and data transfer time of the network are analyzed. Comparisons are made between pipelined and unpipelined (combinational logic) multistage networks.

The partitioning of an interconnection network into independent subnetworks is discussed in Chapter V. This allows a single set of processors to act as many independent SIMD machines. The partitioning properties of single stage, multistage, and pipelined networks are analyzed. The capabilities and restrictions imposed by partitioning are investigated.

In Chapter VI, an enhanced network, the Augmented Data Manipulator, is analyzed. The Augmented Data Manipulator is a highly flexible multistage network. Its capabilities are compared with other networks, and some group theoretic properties of the way in which it passes data are presented.

Image processing tasks can efficiently utilize parallel computer systems. Chapter VII presents three parallel image processing algorithms, a smoothing algorithm, a histogram formation algorithm, and a data classification algorithm. For each, the results of Chapter IV and V are used to analyze the effect of the interconnection network of the parallel system upon the performance of the algorithm.

Little is known about the interaction of interconnection networks and parallel algorithms. An effective system design aid would be one which simulates the effects of a proposed interconnection network. Such a tool, the emulator system, is introduced in Chapter VIII. Consisting of a set of processing elements which interface to a powerful set of interconnections among the processing elements, the emulator system can simulate a wide variety of existing and proposed interconnection networks. The processing elements offer computation capability to test schemes to control an interconnection network.

In this research, the interconnection networks presented in Chapters II and III are studied. The various analysis techniques which are used for this work can be generalized and applied to other interconnection networks. Thus, the significance of this work lies not only in the specific results, but also in the methods used to obtain them.

II. DEFINITIONS

II.1. SIMD Computers

Typically, an SIMD machine (single instruction stream - multiple data stream) [FL66] consists of a control unit (CU), N processors, N memories, and an interconnection network. The CU broadcasts instructions to the N processors, and all active processors execute the same instruction at the same time, but on different data streams. Processors pass data among themselves through the interconnection network. The model for an SIMD computer used here allows each processor a private memory. This combination is referred to as a processing element or PE. The interconnection network links PEs, as shown in Figure II.1, and this model is referred to as the PE-to-PE model. The Illiac IV [BAR68] is configured in this fashion. Another model (Figure II.2), the processor-to-memory model, uses the interconnection network to move data from the processors to the memory and vice versa. The processors and memories of the STARAN computer [BA75] are connected is this manner. The PE-to-PE model can simulate the processor-to-memory model and vice versa. If processor P(i) addresses memory M(j) in the processor-to-memory model, then in the PE-to-PE model, if PE(i) passes an address to PE(j) and PE(j) passes the data from its memory back to

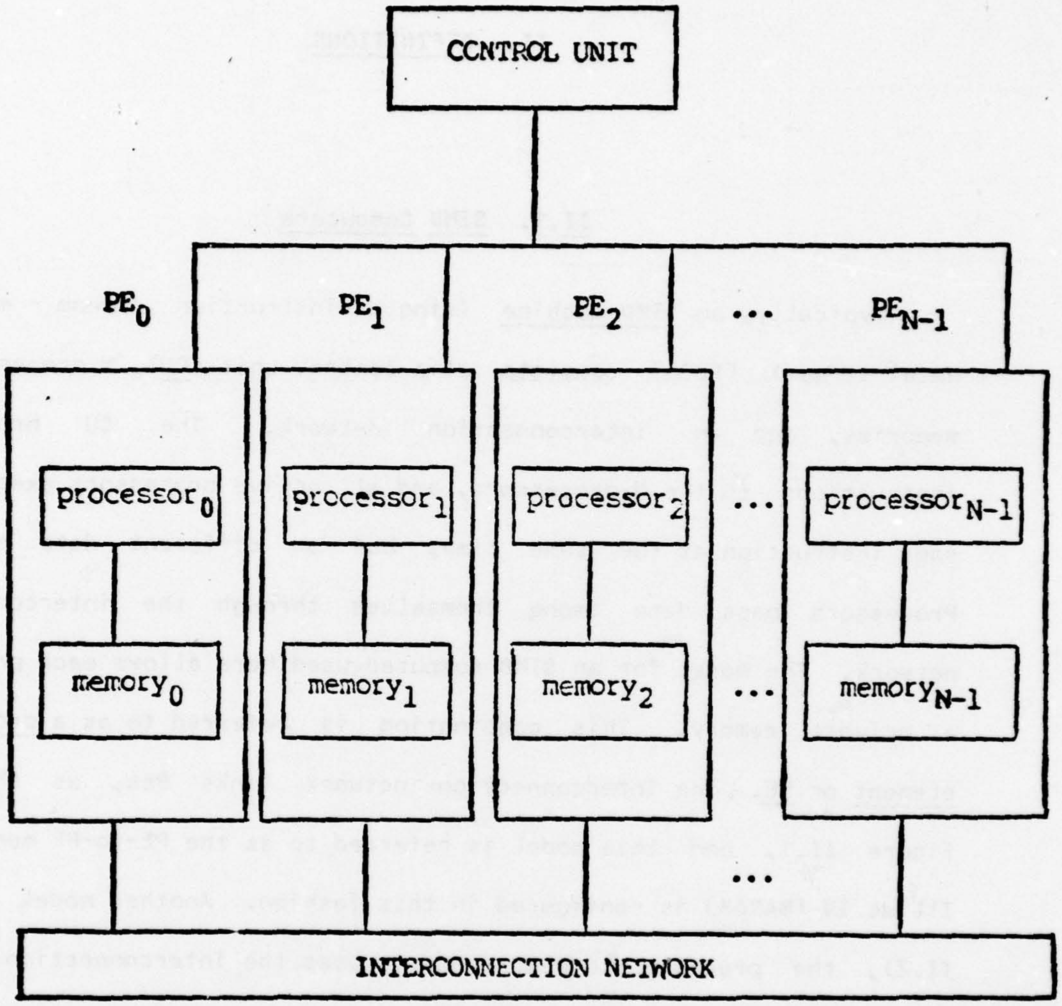


Figure II.1: A PE-to-PE model of an SIMD machine for $N = 2^n$.

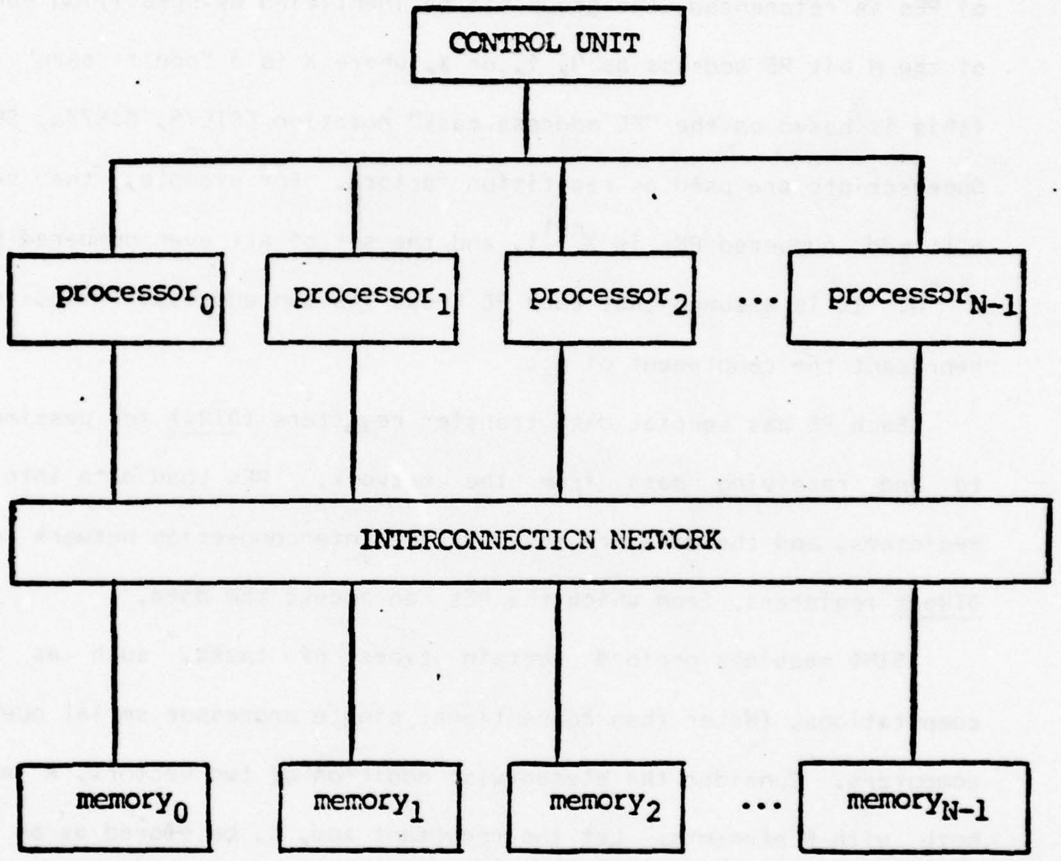


Figure II.2: A processor-to-memory model of an SIMD machine for $N = 2^n$.

PE(i), the effect is the same.

Each PE is assigned a unique address from 0 to N-1, where N is a power of two, that is, $\underline{N} = 2^n$, and $\underline{n} = \log_2 N$. The address in binary of an arbitrary PE P is denoted by $\underline{p}_{n-1}\underline{p}_{n-2}\cdots\underline{p}_1\underline{p}_0$. When a specific group of PEs is referenced, the group can be identified by specifying each bit of the n bit PE address as 0, 1, or X, where X is a "don't care" state (this is based on the "PE address mask" notation [SIE75, SIE77a, SM78]). Superscripts are used as repetition factors. For example, the set of all odd numbered PEs is $X^{n-1}1$, and the set of all even numbered PEs is $X^{n-1}0$. It is assumed that each PE knows its own address. Also, let $\overline{p}_i$ represent the complement of p_i.

Each PE has special data transfer registers (DTRs) for passing data to and receiving data from the network. PEs load data into DTRin registers, and the data are moved by the interconnection network to the DTRout registers, from which the PEs can access the data.

SIMD machines perform certain types of tasks, such as matrix computations, faster than conventional single processor serial operation computers. Consider the elementwise addition of two vectors, A and B, both with N elements. Let the resultant sum, C, be stored as an N word vector. Assume the SIMD machine has a PE-to-PE configuration, and that A(i), B(i), and C(i) are stored in PE(i), $0 \leq i < N$. To compute C, a serial computer executes the code

```
for i = 0 until N-1 step +1 do
    C(i) = A(i) + B(i),
```

and uses N steps to complete the operation. The SIMD computer is a parallel processor, and earns this name by processing all N elements of the vector addition in parallel. So, PE(i) performs $C(i) = A(i) + B(i)$, simultaneously for all i, $0 \leq i < N$. The SIMD computer completes the operation in one step consisting of reading A(i) and B(i) from memory, adding the two, and writing the result into C(i).

In the example above, the SIMD machine completed the task faster than the serial processor because the data were distributed among N PEs, and no communication was needed among the PEs. For some tasks, PEs must pass data among themselves, and it is the interconnection network which supports this data movement.

The next example illustrates the function of the interconnection network of the SIMD machine. Suppose that the vectors A, B, and C are stored as in the previous example, and that each is N words long. Consider the code

```
for i = 1 to N-1 step +1 do
    C(i) = A(i-1) + B(i)
C(0) = B(0).
```

The SIMD computer performs this task in five steps.

(1) Before PE(i) can perform the addition, it must receive A(i-1) from PE(i-1). So, first, PEs 0 through N-2 move data into DTRin simultaneously,

(2) Next, the interconnection network is set to move data from PE(i-1) to PE(i), for all i, $1 \leq i < N$, simultaneously,

(3) Then, PE(i) retrieves the data from DTRout, for all i, $0 \leq i < N$, simultaneously,

(4) The addition is done in PEs 1 through N-1 simultaneously.

(5) Lastly, PE(0) stores B(0) in C(0).

For comparison, the serial processor performs this task in N steps, each of which is a read-add-write step or a read-write step.

These two examples show that, while the SIMD machine has about N times the hardware of a serial processor, it does not always perform a task N times faster. In the second example, the overhead introduced by data transfer among PEs limited the speedup of the task. A more extensive tutorial on parallel processing is found in [KUC77].

II.2. Interconnection Networks

The interconnection network may take many forms. A bus structure (Figure II.3) requires the least hardware of any method. But, only one PE at a time may use the bus, and so transfers that require all PEs to move data are time consuming. At the opposite extreme, a crossbar switch matrix (Figure II.4) can connect any PE to any other PE and can allow all PEs to transfer data simultaneously. Since a switch is required at each switching node of the crossbar, $O(N^2)$ gates are required. Thus, the network is too expensive for use with a large number of processors. Benes [BE65] proposed the rearrangeable switching network, which has the same capability as the crossbar but uses only $O(N \log_2 N)$ gates. But the fastest algorithm to set up the network requires time $O(N \log_2 N)$ [OPT71].

A practical interconnection network must compromise the speed of the crossbar and the cost of the bus. This work will consider networks that are less complex than the crossbar but faster than the bus.

An _interconnection network_ can be described as a set of interconnection functions, where each _interconnection function_ is a permutation (bijection) on the set of PE addresses [SIE75, SIE77a]. When interconnection function f is applied, PE(i), if active, passes its data to PE(f(i)) for all i, $0 \leq i < N$, simultaneously. To pass data from one PE to another PE, a programmed sequence of interconnection functions must be executed. An equivalent definition is that the interconnection network takes the set of PE addresses as its input and produces as its output a permutation of these PE addresses, i.e., it transforms (or maps) input address I to output address O. For example, suppose that

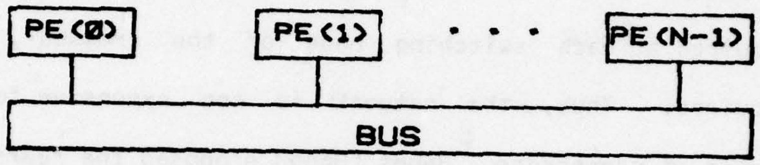


Figure II.3: A bus structure for connecting N PEs.

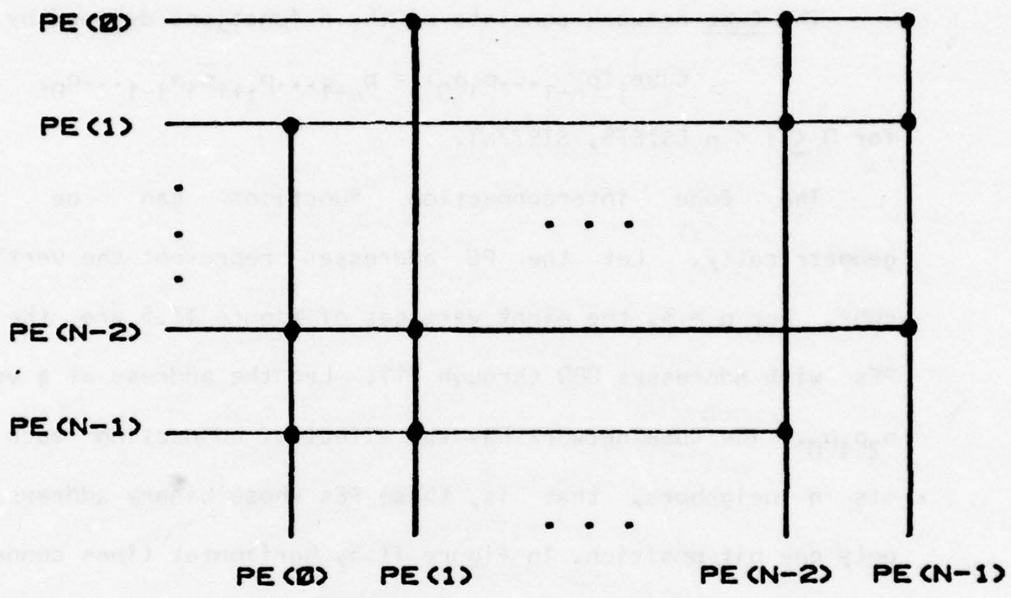


Figure II.4: A crossbar matrix switch for connecting N PEs.

PE(i) wishes to send data to PE(i+1). The resulting permutation is f(i) = (i + 1) modulo N, where i is the address of the PE at the input of the network, and f(i) is the address of the PE that receives data at the output of the network. These two definitions will be used interchangeably.

Four interconnection networks are of particular interest here.

The Cube network consists of the n functions defined by

$$\mathrm{Cube}_i(p_{n-1}\ldots p_1p_0) = p_{n-1}\ldots p_{i+1}\overline{p}_ip_{i-1}\ldots p_0,$$

for $0 \leq i < n$ [SIE75, SIE77a].

The Cube interconnection functions can be interpreted geometrically. Let the PE addresses represent the vertices of an n cube. For n = 3, the eight vertices of Figure II.5 are the the eight PEs with addresses 000 through 111. Let the address at a vertex be $P = p_2p_1p_0$. The Cube network has the effect of connecting each vertex to its n neighbors, that is, those PEs whose binary addresses differ in only one bit position. In Figure II.5, horizontal lines connect vertices whose labels differ in bit p_0, diagonal lines connect vertices whose labels differ in bit p_1, and vertical lines connect vertices whose labels differ in bit p_2. For example, Cube_0 connects the following pairs of processors: 000 and 001, 010 and 011, 100 and 101, and 110 and 111. That is, $\mathrm{Cube}_0(0) = 1$, $\mathrm{Cube}_0(1) = 0$, $\mathrm{Cube}_0(2) = 3$, $\mathrm{Cube}_0(3) = 2$, $\mathrm{Cube}_0(4) = 5$, $\mathrm{Cube}_0(5) = 4$, $\mathrm{Cube}_0(6) = 7$, and $\mathrm{Cube}_0(7) = 6$.

Various types of cube networks have been explored. The multistage network used in the STARAN is a hardware series of cube functions [BA76]. The SW Banyon with S = F = 2 [GL73, GOK76] is a cube type network. The delta networks proposed in [PAT79] include the Cube

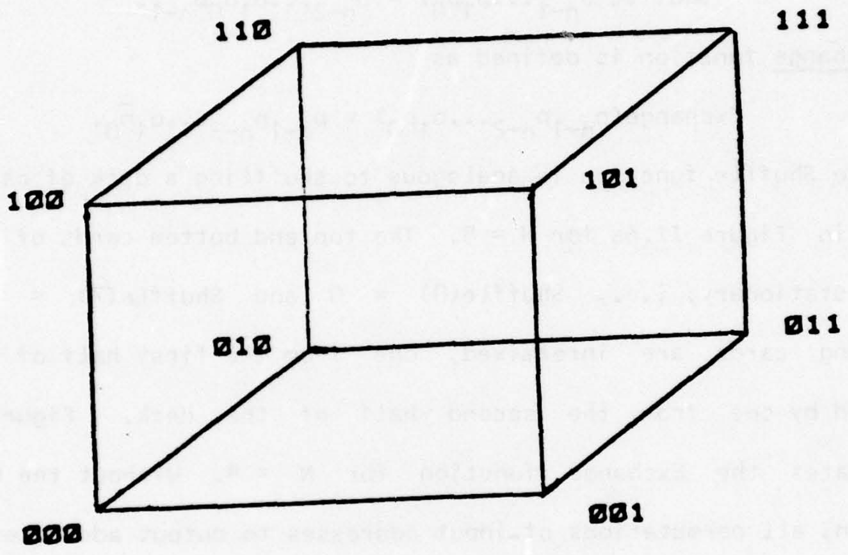


Figure II.5: For N = 8, the Cube interconnection functions can be viewed geometrically as connecting the 8 vertices of a 3 dimensional cube.

topology. The interconnection network for the CHOPP multiprocessor system [SUB77] employs the cube interconnection functions. In [BA76, BAU74, PEA77, SIE79b, SIE78b], the usefulness of this type of network is shown.

The Shuffle-Exchange network [ST71] consists of two functions. The Shuffle function is defined as

$$\text{Shuffle}(p_{n-1}\cdots p_1 p_0) = p_{n-2}\cdots p_1 p_0 p_{n-1}.$$

The Exchange function is defined as

$$\text{Exchange}(p_{n-1}p_{n-2}\cdots p_1 p_0) = p_{n-1}p_{n-2}\cdots p_1\overline{p}_0.$$

The Shuffle function is analogous to shuffling a deck of cards, as shown in Figure II.6a for N = 8. The top and bottom cards of the deck remain stationary, i.e., Shuffle(0) = 0 and Shuffle(7) = 7. The remaining cards are intermixed, one from the first half of the deck followed by one from the second half of the deck. Figure II.6b illustrates the Exchange function for N = 8. Without the Exchange function, all permutations of input addresses to output addresses which the Shuffle-Exchange network could form would require that PEs 0 and N-1 be mapped to themselves. Note that Exchange (P) = Cube_0 (P).

This network is the basis of Lawrie's omega network [LAW75]. It is also included in the networks of the RAP [CGY74] and Omen [HIG72] systems. It has been shown to be useful in [GOL61, LAN76, LAST76, SIE79b, SIE78b, ST71].

The Plus-Minus 2^i (PM2I) network consists of the 2n functions defined by

$$\text{PM2}_{+i}(j) = j + 2^i \text{ modulo } N$$
$$\text{PM2}_{-i}(j) = j - 2^i \text{ modulo } N$$

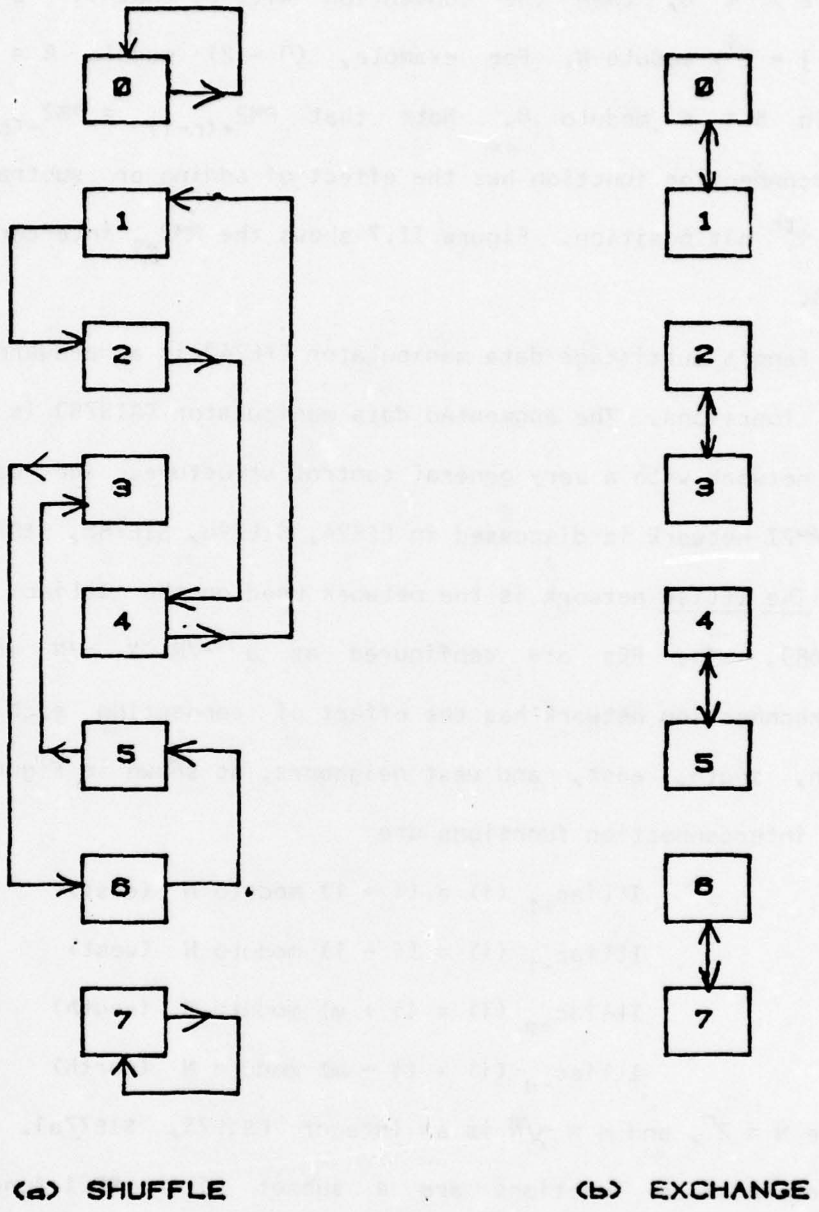


Figure II.6: The Shuffle and Exchange functions for N = 8.

for $0 \leq j < N$, $0 \leq i < n$ [SIE75, SIE77a]. Throughout this discussion, if $(j - 2^i) < 0$, then the convention will be that $(j - 2^i)$ modulo $N = (N + j - 2^i)$ modulo N. For example, $(0 - 2)$ modulo $8 = (8 + 0 - 2)$ modulo $8 = 6$ modulo 8. Note that $PM2_{+(n-1)} = PM2_{-(n-1)}$. A PM2I interconnection function has the effect of adding or subtracting 1 in the i^{th} bit position. Figure II.7 shows the $PM2_{\pm 1}$ interconnections for $N = 8$.

Feng's multistage data manipulator [FE74] is a hardware series of PM2I functions. The augmented data manipulator [SIS78] is a multistage PM2I network with a very general control structure. The usefulness of the PM2I network is discussed in [FE74, SIE79b, SIE78b, SIS78].

The Illiac network is the network used on the Illiac IV computer [BAR68]. The PEs are configured as a $\sqrt{N} \times \sqrt{N}$ array, and the interconnection network has the effect of connecting each PE to its north, south, east, and west neighbors, as shown in Figure II.8. The four interconnection functions are

$$\text{Illiac}_{+1}(i) = (i + 1) \text{ modulo } N \quad \text{(east)}$$

$$\text{Illiac}_{-1}(i) = (i - 1) \text{ modulo } N \quad \text{(west)}$$

$$\text{Illiac}_{+m}(i) = (i + m) \text{ modulo } N \quad \text{(south)}$$

$$\text{Illiac}_{-m}(i) = (i - m) \text{ modulo } N \quad \text{(north)}$$

where $N = 2^n$, and $m = \sqrt{N}$ is an integer [SIE75, SIE77a]. The Illiac interconnection functions are a subset of the PM2I functions, where $\text{Illiac}_{+1}(i) = PM2_{+0}(i)$, $\text{Illiac}_{-1}(i) = PM2_{-0}(i)$, $\text{Illiac}_{+m}(i) = PM2_{+n/2}(i)$, and $\text{Illiac}_{-m}(i) = PM2_{-n/2}(i)$.

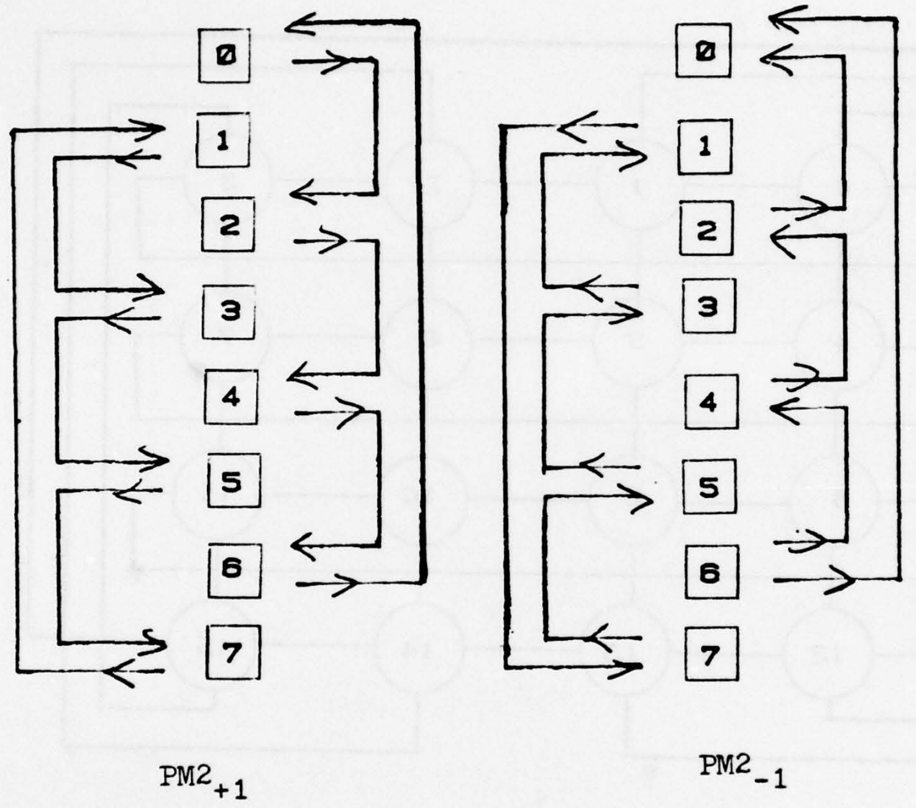


Figure II.7: The $PM2_{+1}$ interconnection functions connect PE(i) to PE(i+2 modulo N), and the $PM2_{-i}$ interconnection functions connect PE(i) to PE(i-2 modulo N), $0 \leq i < N$. Here, $N = 8$.

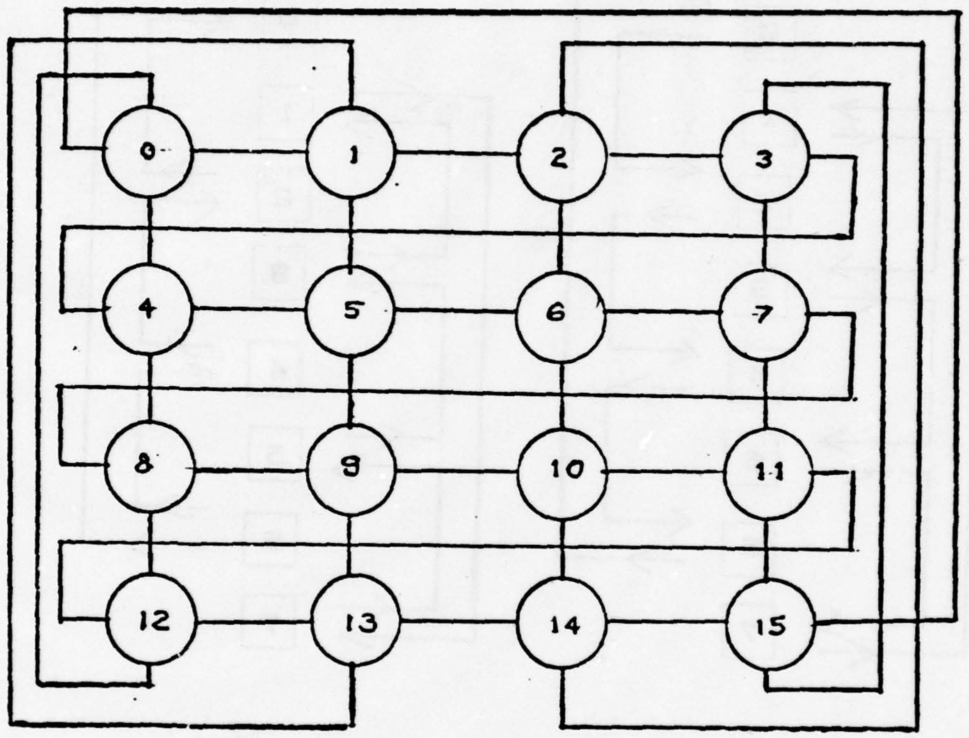


Figure II.8: For N = 16, the Illiac network connects PE(i) to PE(i+1 modulo 16), to PE(i-1 modulo 16), to PE(i+4 modulo 16), and to PE(i-4 modulo 16).

II.3. Network Structures

A network can be constructed as either a recirculating or a multistage network. A recirculating network is an interconnection network with a single stage of switches. The stage is reused until data reach their final destinations. A complete data transfer may take several passes through the network. Figure II.9 illustrates this arrangement. A multistage network is an interconnection network composed of several, usually $\log_2 N$, stages of combinational logic switches. In general, a single pass through a multistage network is sufficient to route data to their destinations. However, when a single pass is insufficient, multiple passes may be used. Figure II.10 illustrates a multistage network.

For constructing multistage networks such as the STARAN and omega networks, the interchange box is a useful building block [SIS78]. The interchange box is a two-input two-output device that, in the most general case, may assume one of four legitimate states (Figure II.11). Let the upper input and output lines be labeled i and the lower input and output lines be labeled j. The four legitimate states are: (1) straight - input i to output i, input j to output j; (2) exchange - input i to output j, input j to output i; (3) lower broadcast - input j to outputs i and j; (4) upper broadcast - input i to outputs i and j [LAW75]. A two function interchange box is defined to be an interchange box capable of either the straight or exchange states. A four function interchange box is defined to be an interchange box capable of being in any of the four legitimate states.

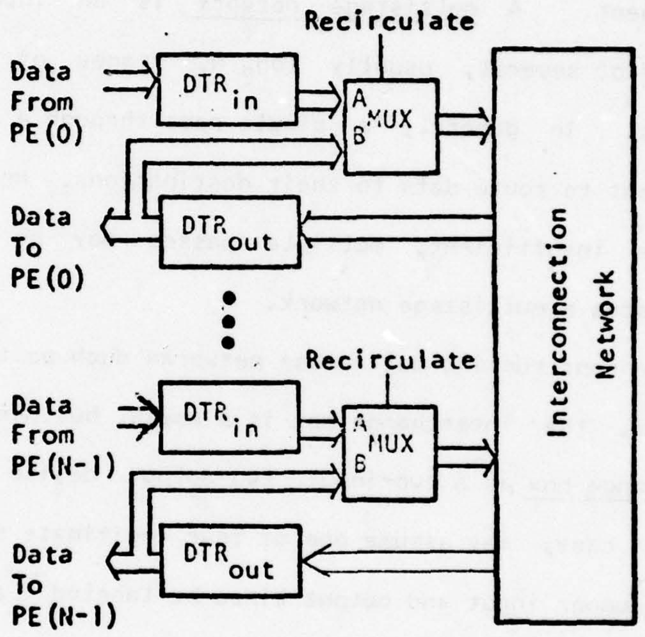


Figure II.9: A model for a recirculating network for PE(i).

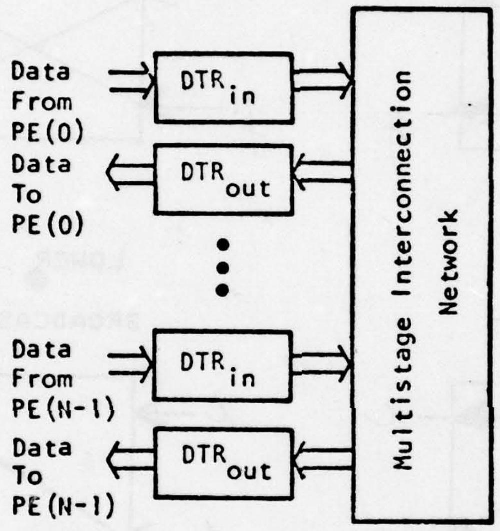


Figure II.10: A model for a multistage interconnection network for PE(i).

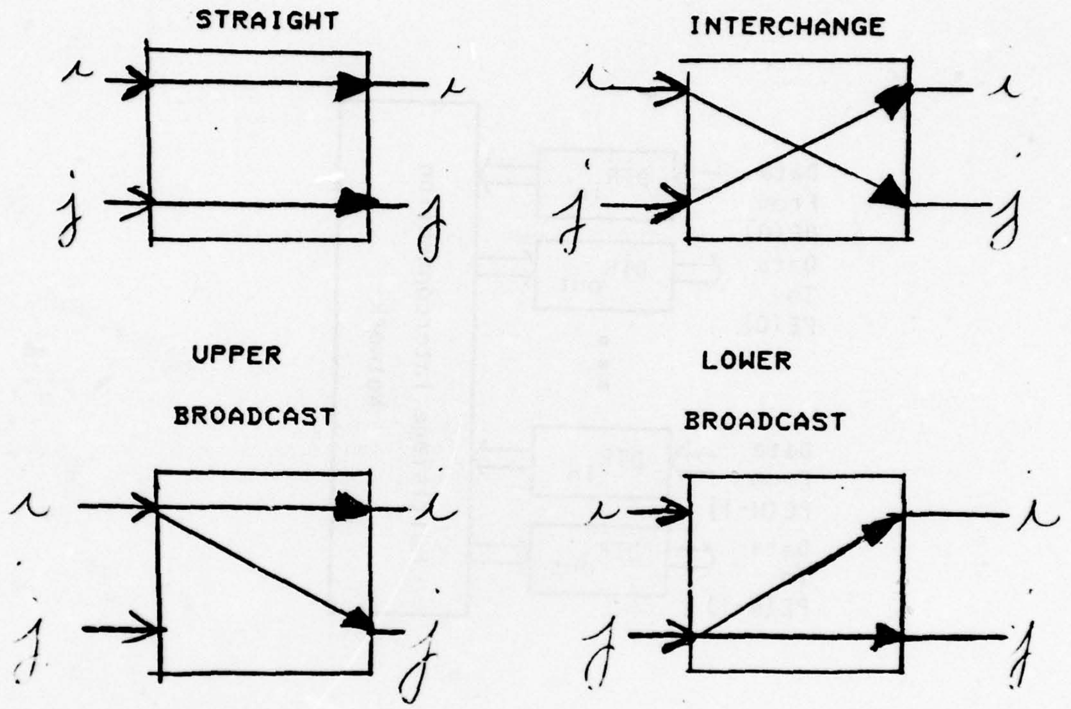


Figure II.11: The interchange box is a two-input two-output device that can be in one of four legitimate states: straight, exchange, lower broadcast, or upper broadcast.

The control structure of a network is an important consideration. For multistage networks, three types of controls are discussed in [SIS78]. Individual stage control allows one control signal for each stage of the network. Individual box control uses a separate control signal for each interchange box in the network, using hardware [PEA77] or software (destination tags) [LAW75]. Partial stage control uses more than one but less than N/2 control signals at any stage of the network.

The typical control mechanism for a recirculating network assumes that only an active PE can send and receive data. An inactive PE can only receive data, because an interprocessor data transfer instruction is executed only by active PEs. Here, a control is introduced that differs from the usual SIMD control, where there is a single instruction stream, and all active PEs must execute the same interconnection function. By providing each PE with its own routing control register, this restriction is removed. Independent function control allows each PE to execute any set of the implemented interconnection functions. For example, using a Cube network, PE(P) might send data to all PEs whose addresses differ in one bit from P's address by executing all $\log_2 N$ Cube functions, Cube_0, Cube_1, etc. Also, different PEs may use different functions, e. g., PE(0) may use Cube_0 while PE(1) uses Cube_1.

II.4. PE Address Masks

In the normal execution of an SIMD program, all PEs will respond to the instructions issued by the control unit. A masking scheme can be provided which allows the user to select a subset of PEs to respond to the instructions. To select which PEs are active, an n position PE address mask may accompany an instruction [SIE75, SIE77a]. Recall the address specification notation of section II.1. Each position of the mask can be 0, 1, or X (don't care). For a given mask, the PEs whose addresses match the mask are active. For example, if N=8 and the mask specified is

MASK [01X]

then the active PEs are 010 and 011, and only these two respond to the instruction which follows the MASK command.

A negative PE address mask is the same as a regular PE address mask, except that it activates all those processors which do not match the mask [SM78]. This type of mask can activate sets of processors that a single regular mask cannot. A negative PE address mask is prefixed by a '-'. Superscripts are used as repetition factors when describing masks. For example for $N = 2^n$, the command

MASK $[-0^n]$

activates all PEs except PE(0).

Logical operations can be applied to two PE address masks to specify another set of PEs. The logical OR of two masks forms the union of the two sets of PEs specified by the masks. The logical AND of two masks forms the intersection of the two sets of PEs. For example, the command

$$\text{MASK } [X^{n-1}0] \text{ OR } [X^{n-2}01]$$

activates all PEs with even addresses and all PEs whose addresses end in 01.

Other masking schemes may be used. The general address masks of the Illiac IV computer use a bit vector of length N [BAR68]. PE(i) is active if and only if the i th bit of the vector is one. Data conditional masks reflect "if-then-else" statements [SM78]. When a conditional statement is encountered in a program, each PE executes the statement for different data, and so the outcome may be different from one PE to the next. Consequently, each PE sets an internal flag so that it will be active for either the "then" or the "else" but not both. So, each PE is conditionally active based on the results of a comparison.

In describing the simulation algorithms of Chapter VIII, an Algol-like language will be used. It includes statements that indicate which PEs are to be active during execution of an algorithm. "for all PEs" means that all N PEs are to execute the code which follows. "if A then B" statements first cause A to be evaluated. Only PEs for which A is true are active for the execution of B; all others are inactive. "if A then B else C" statements cause A to be evaluated, disable PEs for which A is false, execute B, disable PEs where A is true and enable PEs for which A is false, and then execute C [SM78].

III. LITERATURE REVIEW

The traditional N X N crossbar switch is too expensive for use in a large SIMD computer. Other networks have been proposed that can produce all permutations of PE addresses in one pass through the network.

One such network is the rearrangeable switching network [BE65], which uses $O(N \log_2 N)$ gates, but requires $O(N \log_2 N)$ time to set up the network [OPT71]. Feierbach and Stevenson [FS77] have investigated this network for use in an SIMD computer with 1024 PEs. Algorithms are presented which implement a k-shift (PE(j) sends data to PE(k+j)), the perfect shuffle, and broadcasting (PE(j) sends data to all other PEs). Figure III.1 shows this network built using two function interchange boxes for N = 8.

Batcher's sorting network [BA68, KN73] could be used as an interconnection network. It requires $O(N (\log_2 N)^2)$ gates, and requires $O((\log_2 N)^2)$ time to pass data through the network. This network is shown for N = 8 in Figure III.2 [SIE78b]. The building block for this network compares its two inputs and orders them accordingly at the outputs.

Both recirculating and multistage Shuffle-Exchange networks have been examined in the literature.

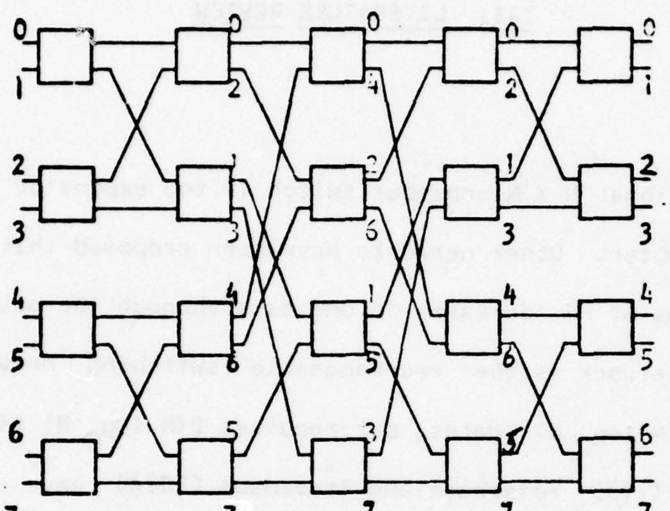
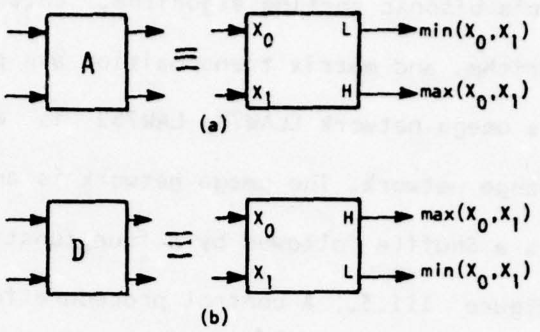
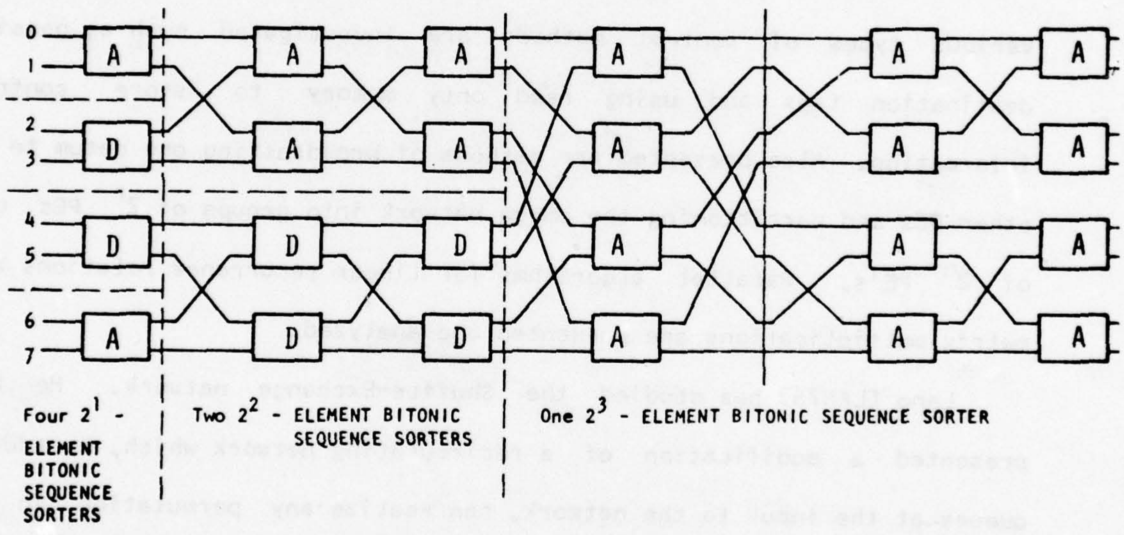


Figure III.1: The programmable switching network [FS77] is a Benes rearrangeable switching network, shown here for N = 8.



(a) Ascending comparator.
(b) Descending comparator.



0
1
2
3
4
5
6
7
A
D
D
A
Four 2^1 - ELEMENT BITONIC SEQUENCE SORTERS
Two 2^2 - ELEMENT BITONIC SEQUENCE SORTERS
One 2^3 - ELEMENT BITONIC SEQUENCE SORTER

Figure III.2: A bitonic sorter for an arbitrary sequence of elements [SIE78b].

Stone [ST71] has published algorithms which show how the perfect shuffle can be used. Algorithms for polynomial evaluation, sorting using Batcher's bitonic sorting algorithm, calculating the FFT using Pease's algorithm, and matrix transposition are presented.

Lawrie's omega network [LAW73, LAW75] is an expanded multistage Shuffle-Exchange network. The omega network is an n stage network, where each stage is a Shuffle followed by a four function interchange box as shown in Figure III.3. A control procedure for the omega network was presented in [LAW75] where destination tags for each datum determine its path through the network.

Wen [WEN76] also has presented results for the omega network. Various types of control methods are investigated such as passing destination tags and using read only memory to store control information. Also presented are methods of broadcasting one datum to 2^k other PEs and partitioning the omega network into groups of 2^r PEs out of 2^n PE's. Parallel algorithms for linear recurrence relations and matrix multiplications are presented and analyzed.

Lang [LAN76] has studied the Shuffle-Exchange network. He has presented a modification of a recirculating network which, by adding queues at the input to the network, can realize any permutation in at worst O(-/N) time. A simplified Shuffle-Exchange network has been presented [LAST76]. This network needs less logic to control the network than the omega network. While it cannot perform all omega network permutations, it can form many useful ones. A method of partitioning this network has been given which assumes that all partitions perform the same interconnection function.

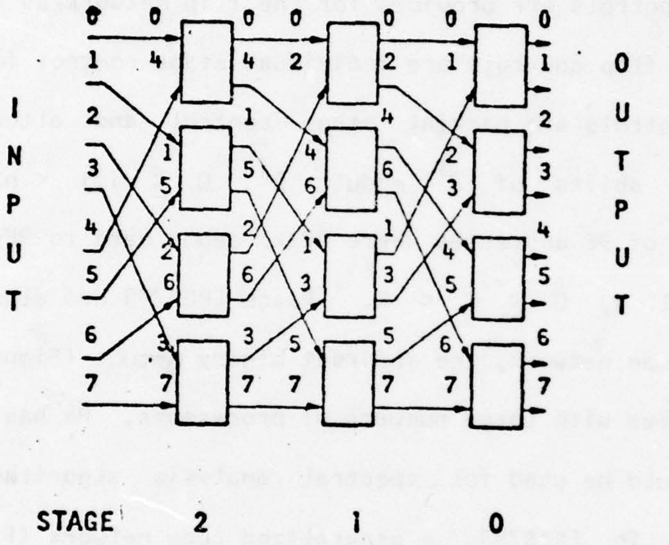


Figure III.3: The omega network is a $\log_2 N$ stage Shuffle-Exchange network with four function interchange boxes [LAW75].

Several Cube networks have been presented in the literature. The CHOPP machine [SUB77], a multiple instruction stream - multiple data stream (MIMD) machine [FL66] design, uses a recirculating cube network to move packets of information among processors. The flip network of the STARAN [BA76] SIMD machine is a multistage Cube network which moves data between processors and memory or between processors and processors. Two set of controls are provided for the flip network as shown in Figure III.4. The flip controls are individual stage control for the network. The shift controls are partial stage control and allow the network perform all shifts of 2^i modulo 2^j, $0 \leq i,j < n$. That is, the permutations of PE addresses where PE(k) sends data to PE(k + 2^i modulo 2^j) for all k, $0 \leq k < N$. Pease [PEA77] has also worked with a multistage Cube network, the indirect binary n-cube (Figure III.5), for use in systems with large numbers of processors. He has shown how such a network could be used for spectral analysis algorithms and matrix operations. In [SIS78], a generalized cube network (Figure III.6), a multistage Cube network, was introduced and used as a basis for comparing multistage Cube networks.

Feng's data manipulator [FE74] is based on the PM2I functions. The data manipulator network (Figure III.7) consists of n stages of N cells, where for $0 \leq j < N$, and $0 \leq i < n$, there are three sets of interconnections from input cell j at stage i: $PM2_{+i}$, $PM2_{-i}$, and straight to output cell j. Each stage of the network is controlled by a pair of signals selected from a group of six. U_1 ($PM2_{-i}$), D_1 ($PM2_{+i}$), and H_1 (straight) control cells whose i^{th} bit of the address is 0, and

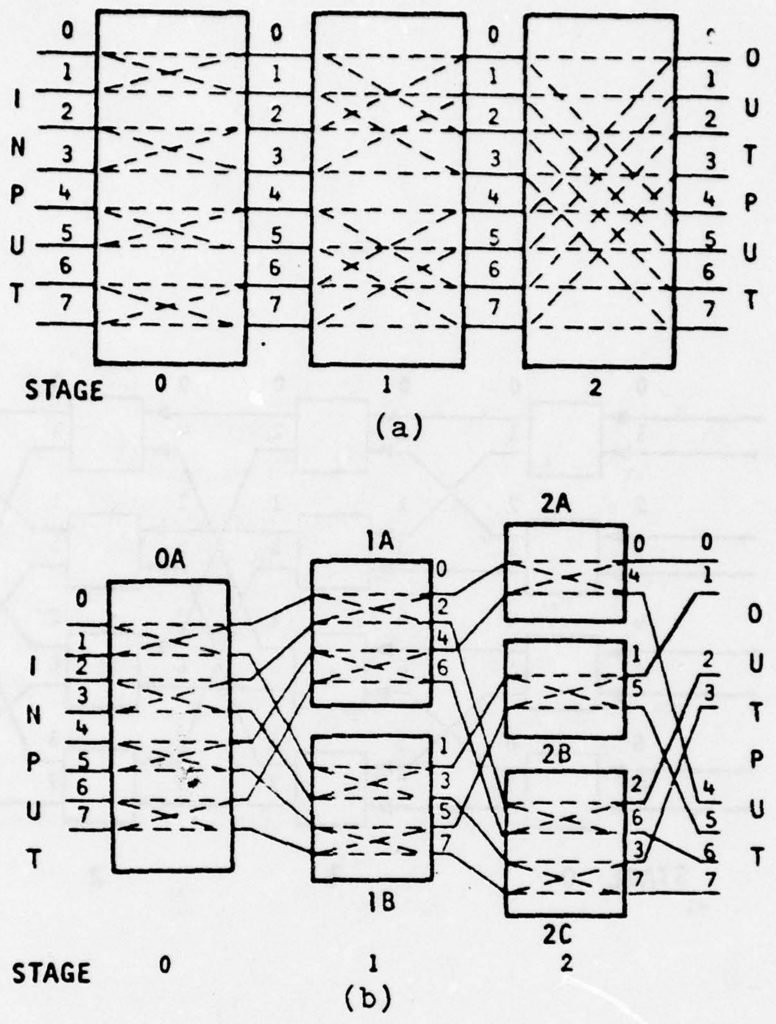


Figure III.4: The STARAN flip network for N = 8 [BA76]. (a) The flip control is individual stage control. (b) The shift control is partial stage control.

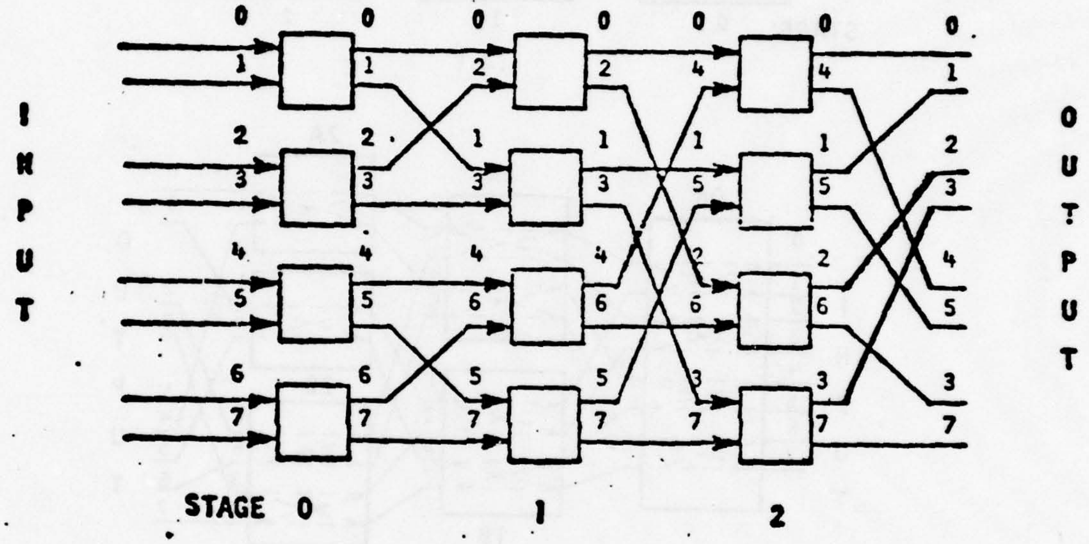


Figure III.5: The indirect binary n-cube is a $\log_2 N$ stage cube network, shown here for N = 8. The first stage forms Cube_0, the second Cube_1, and the last Cube_2 [PEA77].

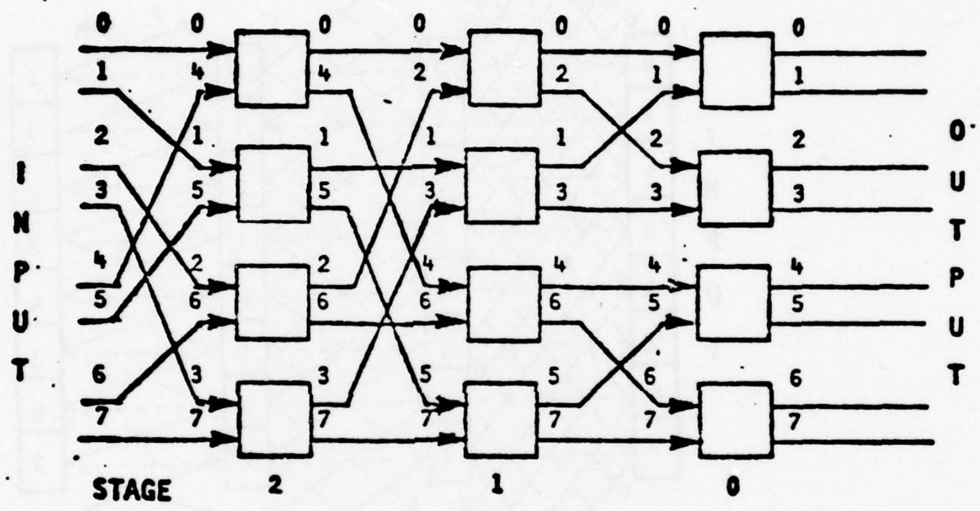


Figure III.6: The generalized cube network, shown here for N = 8, is a $\log_2 N$ stage cube network. The first stage, stage 2, forms Cube_2, the next forms Cube_1, and the last forms Cube_0.

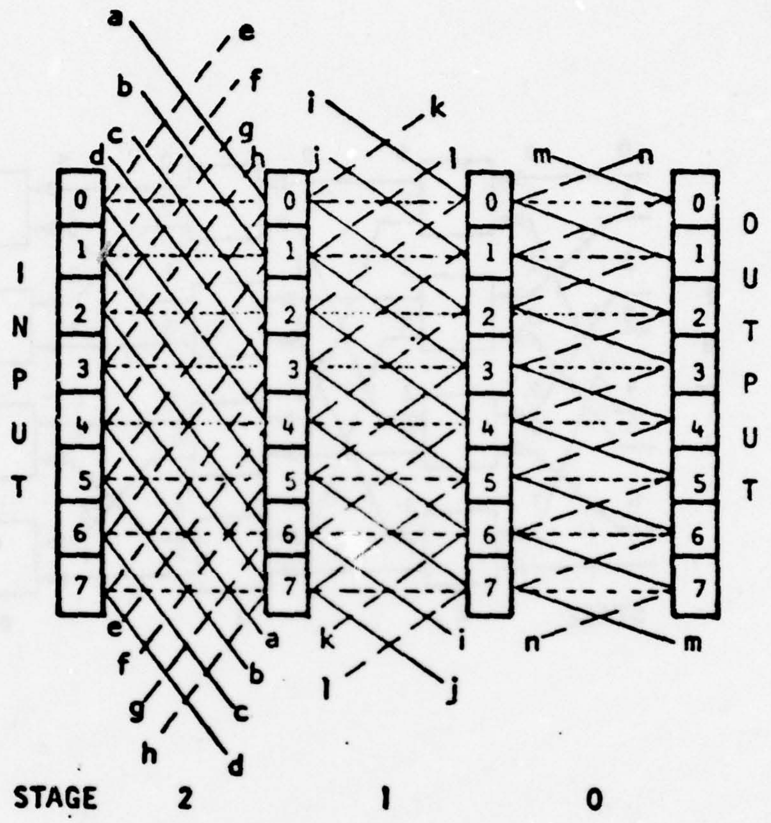


Figure III.7: The data manipulator network [FE74], shown for N = 8, is a $\log_2 N$ PM2I network.
The dashed lines represent the U control line interconnections.
The dotted lines represent the H control line interconnections.
The solid lines represent the D control line interconnections.
For stage i, U_1, D_1, and H_1 control those cells whose ith bit is 0, and U_1, D_2, and H_2 control those cells whose ith bit is 1.

U_2 ($PM2_{-i}$), D_2 ($PM2_{+i}$), and H_2 (straight) control those cells whose i^{th} bit is 1.

The augmented data manipulator [SIS78, SIE79a] is a data manipulator with individual cell control. That is, each cell receives none, one, or two of the signals H, U, and D. Since each cell in controlled independently, the set of permutations that the network can perform is a superset of those of the generalized cube network [SIS78].

Siegel has presented comparisons of the Shuffle-Exchange, Cube, PM2I, and Illiac networks [SIE77b]. Lower and upper time bounds have been presented for each network to simulate any other [SIE77a, SIE79b]. The effects of PE address masks have been related to the number of permutations on the set of PE addresses that a network can perform [SIE77a]. Algorithms have been presented which show how a network can simulate an arbitrary interconnection with the aid of Batcher's bitonic sorting algorithm by sorting destination tags associated with the data presented to the network [SIE78b]. In [SIS78, SIE79a], it was shown that some multistage networks have the same topologies and so can perform the same permutations. This fact will be used extensively in proving theorems in Chapters V and VIII.

IV. NETWORK STRUCTURES

IV.1. Introduction

Three types of interconnection functions, the Cube, the Shuffle-Exchange, and the Plus-Minus 2^i, are implemented as recirculating (single stage) networks and as multistage combinational logic networks in section 2. Comparisons are made on hardware complexity and delay to transfer data. The Shuffle-No Shuffle-Exchange network is introduced in section 3. This network has all the capabilities of the multistage Shuffle-Exchange network, but in addition, can perform 1 to n-1 shuffles in one pass through the network. Section 4 considers breaking a data word into segments before passing the datum through the network. Then, the width of the network is smaller than when the wider data word is passed all at once. So, more than one pass is made through the network to pass the data. Alternatively, for a multistage network, the network can be pipelined and the S segments passed in parallel. The cost and delay to pass S segments are compared for these cases.

IV.2. Hardware Implementations

In order to compare these networks, typical circuits for each are presented. Comparisons of gate count and circuit delay are made for multistage and recirculating Shuffle-Exchange, Cube, and PM2I networks. For simplicity, the following analysis is made on networks that are one bit wide. The costs of DTRin, DTRout, and any hardware needed to interface with the network are not included in the network cost estimates. The delay times presented are intended as approximations and are useful for comparisons. In practice, the speed of the network will also depend on the speed at which control signals can be generated and on the technology of the circuit design. Table IV.1 summarizes the notation that will be used in the discussion.

Three multistage Cube networks which have been presented in the literature are the STARAN flip network [BA76], the indirect binary n-cube [PEA77], and the generalized cube [SIS78]. In [SIS78], it was shown that these three networks and an n-stage Shuffle-Exchange network are all topologically equivalent.

With this in mind, a circuit for an 8-item generalized cube network is presented in Figure III.6. At the input and output to each stage, each line has an n bit binary address, P, $0 \leq P < N$. Stage i compares input lines l and l', whose addresses differ in only bit i, and conditionally exchanges data between l and l'. In this way, a Cube_i function is implemented. The interchange box (Figure IV.1), on which this Cube network can be based, conditionally interchanges the data at its inputs, thus performing a conditional exchange. So, stage i of the network forms Cube_i, $0 \leq i < n$. The circuit uses $n*N/2$ interchange

Table IV.1: Definitions and abbreviations

NOTATION	MEANING
dr	delay of a register
cr	cost of a register
dm	delay of a multiplexer
cm	cost of a multiplexer
dms	delay of the logic for one stage of a multistage network
cms	cost of the logic for one stage of a multistage network
drn	delay of the logic for a recirculating network
drs	delay of the logic for a recirculating Shuffle-Exchange network
cbr	cost of the logic (buffers) for a recirculating network
Cr	cost of a recirculating network
Cp	cost of a pipelined multistage network
Cm	cost of a combinational logic multistage network
Tr	time delay of a recirculating network
Tm	time delay of a combinational logic multistage network
Tp	time delay of an n-stage pipelined multistage network
Tk	time delay of a k-stage pipelined multistage network
W	width of a data word to be transmitted through the network
S	number of segments into which a data word is divided
Q	the number of passes made through a recirculating network to complete a desired transfer
N	the number of PEs in the system
n	$\log_2 N$

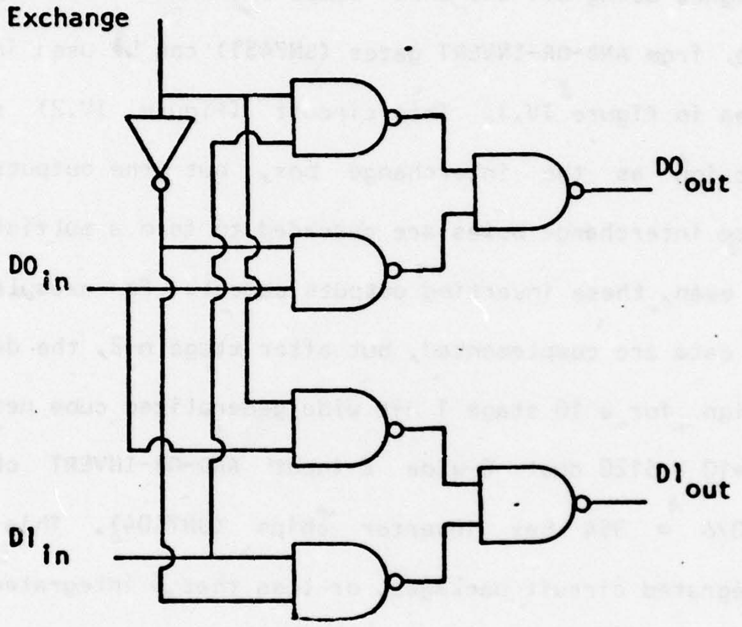


Figure IV.1: An interchange box. If "Exchange" = 0, then DO_{out} = DO_{in} and $D1_{out}$ = $D1_{in}$. If "Exchange" = 1, then DO_{out} = $D1_{in}$ and $D1_{out}$ = DO_{in}.

boxes, or 7n*N/2 gates, and has a delay of dms*n + 2*dr, where dms is the delay through the interchange box, and 2*dr represents the delay through DTRin and DTRout.

As a design example, suppose it is desired to build a multistage generalized cube network for N = 1024. This 10 stage network could be designed using off-the-shelf components. An inverting interchange box made from AND-OR-INVERT gates (SN7451) can be used in place of the NAND gates in Figure IV.1. This circuit (Figure IV.2) performs the same function as the interchange box, but the outputs are complemented. Since interchange boxes are cascaded to form a multistage network, if n is even, these inverting outputs cancel. For example, after stage n-1, the data are complemented, but after stage n-2, the data are true. A design for a 10 stage 1 bit wide generalized cube network would require 512*10 = 5120 dual 2-wide 2-input AND-OR-INVERT chips (SN7451) and 5120/6 = 854 hex inverter chips (SN7404). This is a total of 5974 integrated circuit packages, or less that 6 integrated circuit packages per bit per PE.

Alternatively, an interchange box can be constructed from two 2-line-to-1-line multiplexers. Let the output of multiplexer zero be $D0_{out}$, and let the output of multiplexer one be $D1_{out}$. If exchange = 0, then multiplexer zero selects $D0_{in}$ and multiplexer one selects $D1_{in}$. If exchange = 1, then multiplexer zero selects $D1_{in}$ and multiplexer one selects $D0_{in}$. One integrated circuit (an SN74157) may contain four such multiplexers which all respond to the same control signals. So, a 2-bit wide generalized cube network for N = 1024 can be built using 512*10 = 5120 quadruple 2-line-to-1-line multiplexers (SN74157), or less than 3

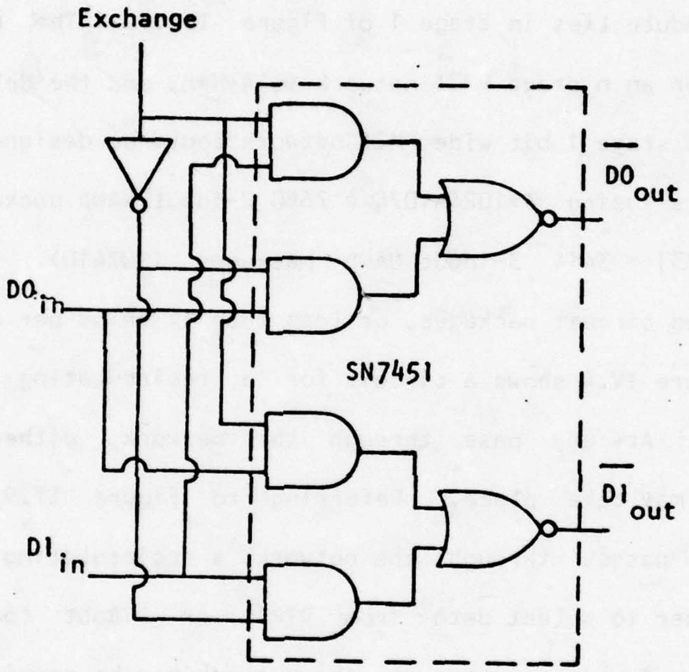


Figure IV.2: An inverting interchange box using AND-OR-INVERT logic.

integrated circuit packages per bit per PE.

A multistage PM2I network can be built from the modules of Figure IV.3 [FE74]. Referring to the multistage network for N=8 in Figure III.7, a logic module of stage 2 has the construction of the left-hand side of the module of Figure IV.3, while the receiving right-hand side of the module lies in stage 1 of Figure III.7. The number of gates needed for an n stage PM2I network is 4*N*n, and the delay is dms*n+2dr.

A 10 stage 1 bit wide PM2I network could be designed from discrete components using 3*1024*10/4 = 7680 2-input NAND packages (SN7400) and |1024*10/3| = 3414 3-input NAND packages (SN7410). This is 11,094 integrated circuit packages, or less than 11 chips per bit per PE.

Figure IV.4 shows a circuit for a recirculating Shuffle-Exchange network. At any pass through the network, either a shuffle or an exchange may take place. Referring to Figure II.9, to accomplish multiple passes through the network, a recirculating network allows a multiplexer to select data from DTRin or DTRout for input to the network. Q passes through the network may be required to complete a desired transfer of data between PEs. This circuit uses 3N gates and has a delay of dr + Q(dr + dm + drs).

The recirculating Cube network can be built using tri-state buffers with outputs that can be OR-TIED together, as shown in Figure IV.5. A recirculating PM2I network could be constructed similarly (Figure IV.6). The Cube network uses N*n buffers while the PM2I network uses 2*N*(n-1) buffers, since $PM2_{+(n-1)} = PM2_{-(n-1)}$. Both have delay dr + Q (dr + dm + drn). For small n, the Cube network could be built using 2 levels of NAND gates, where the input to the network is composed of 2-input NANDs

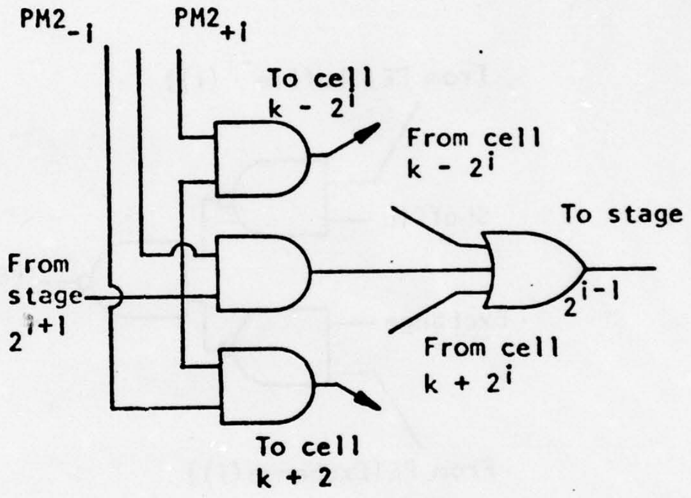


Figure IV.3: A PM2I module for row k [FE74].

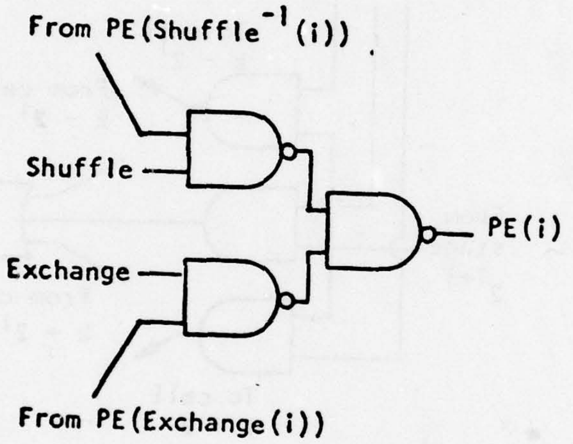
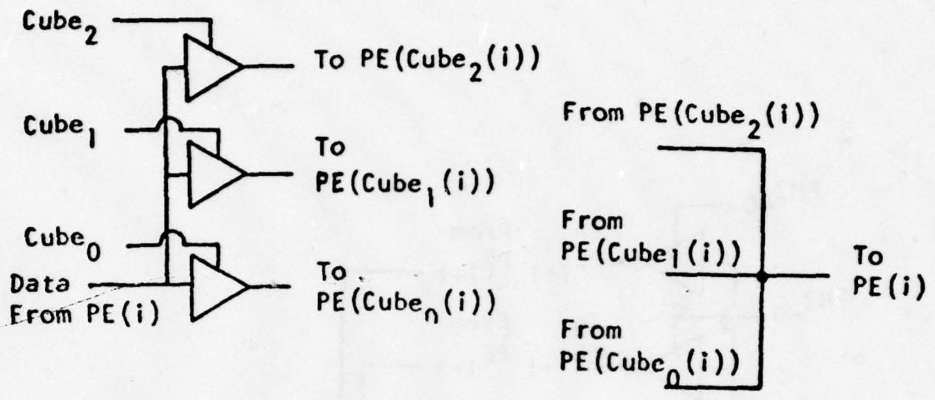


Figure IV.4: A circuit for a recirculating Shuffle-Exchange network for PE(i), $0 \leq i < N$.



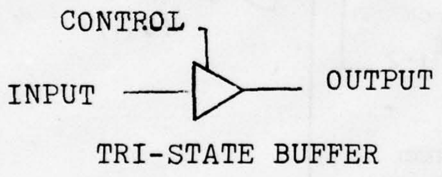


Figure IV.5: A circuit for a recirculating Cube network for PE(i), $0 \leq i < 8$.

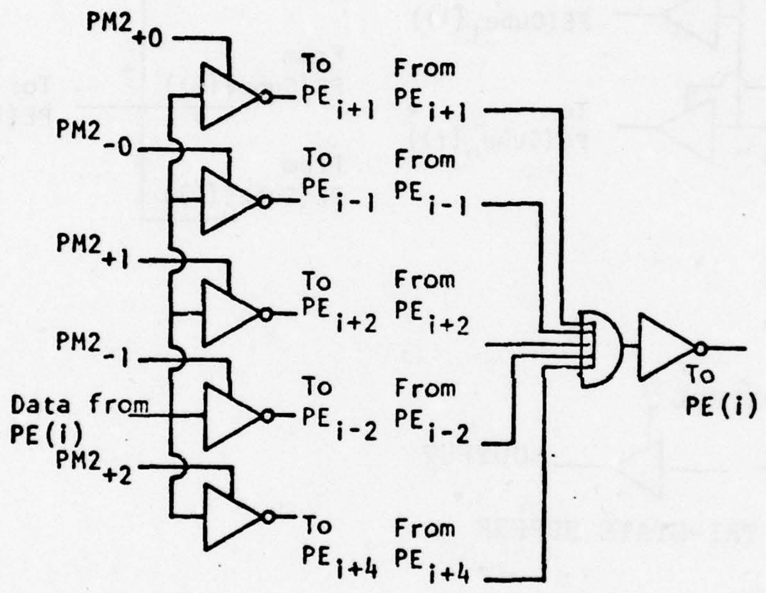


Figure IV.6: A recirculating PM2I network for PE(i), $0 \leq i < 8$.

and the receiving side is composed of n-input NANDS. For large n, due to fan-in and integrated circuit package count limitations, the tri-state buffer design is preferable.

When selecting an interconnection network, the time to complete a data transfer is an important consideration. For a recirculating network, this time is proportional to the number of passes made through the network, Q, $0 \leq Q \leq n$. In the case of the multistage network, for any data transfer, all n stages must be traversed, so as the number of PEs grows, the time to pass data increases.

For some value of Q, a multistage network and a recirculating network will have the same delay time. If dr = 6 ns, dm = 5 ns, drn = 4.5 ns, and dms = 6 ns, then the two networks will require the same delay time if Q = .39 (1 + n). This indicates that the choice between a recirculating network and a multistage network should depend on the number of interconnection functions used on the average to complete a data transfer, and thus depend on the types of problems that a system is designed to perform. For example, if skewed storage [ST75] is used, there will be uniform shifts which may require all n Cube functions of a multistage network. However, sorting using the Cube functions will need only one interconnection function at a time [SIE78b]. Figure IV.7 illustrates this effect for an n-stage generalized cube and a recirculating Cube network.

In Chapter II, three types of controls for multistage networks were defined: individual stage control, individual box control, and partial stage control. The multistage network implementations discussed here can be used with any of these control schemes. The recirculating

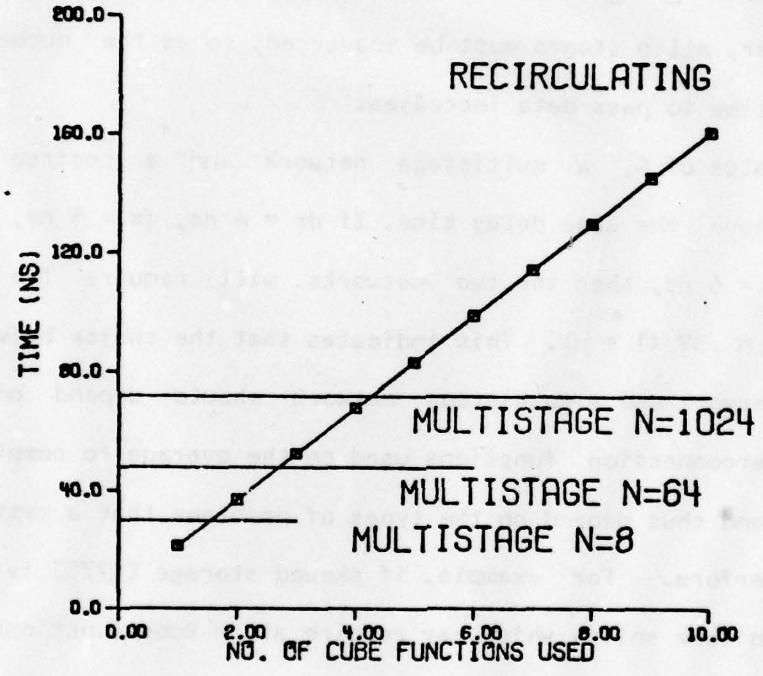


Figure IV.7: Break even points for a multistage Cube network versus a recirculating Cube network.

networks designed here can be used with either the conventional control or the independent function control defined in Chapter II.

IV.3. The Shuffle-No Shuffle-Exchange Network

For the Cube and PM2I networks, any data transfer that can be accomplished in one pass through a recirculating network can be done in one pass through a multistage network. This is not true for the multistage Shuffle-Exchange network. Recall from Chapter II that the shuffle permutation maps network input $P = p_{n-1}..p_1p_0$ to network output

$$\text{Shuffle}(P) = p_{n-2}...p_1p_0p_{n-1}.$$

From Theorem 2 of [LAW75], it can be shown that the multistage Shuffle-Exchange network cannot perform the shuffle permutation in a single pass through the network. Theorem 2 states that for all mappings of source PE Si to destination PE Di that define a permutation of input PEs to output PEs, a $\log_2 N$ stage Shuffle-Exchange network can produce this mapping if and only if

$$Si \neq Sj \Rightarrow$$

$$(\ Si \text{ modulo } 2^k \neq Sj \text{ modulo } 2^k$$

$$\text{OR } \left|\frac{Di}{2^k}\right| \text{ modulo } 2^{n-k} \neq \left|\frac{Dj}{2^k}\right| \text{ modulo } 2^{n-1}),$$

for all k, $1 \leq k \leq n$, and for all i, j, $0 \leq i, j < N$, where $|A|$ is the greatest integer that is less than or equal to A. Consider two source destination pairs that occur in specifying the shuffle permutation:

$$(Si,\ Di) = (0p_{n-2}...p_1p_0,\ p_{n-2}...p_1p_00)$$

$$(Sj,\ Dj) = (1p_{n-2}...p_1p_0,\ p_{n-2}...p_1p_01).$$

Then, $Si \neq Sj$, but Si modulo 2 = Sj modulo 2 and Di/2 modulo 2^{n-1} = Dj/2 modulo 2^{n-1}. For k = 1, the two source destination pairs do not satisfy the criteria of Theorem 2 of [LAW75], and so the multistage Shuffle-Exchange network cannot pass the shuffle permutation.

To rectify this, a Shuffle-No Shuffle-Exchange (SNSE) network is introduced here. At each stage of this multistage network (Figure IV.8), the options are staight (do nothing), Shuffle, Exchange, or Shuffle-Exchange. Data passes through stage n-1, ..., 1, and last passes through stage 0. Alternatively, the cell i of the network can be designed as a multiplexer which conditionally outputs the data input to cell i, data from $\text{Shuffle}^{-1}(i)$, data from Exchange(i), or data from $\text{Shuffle}^{-1}(\text{Exchange}(i))$. Assume that at any stage, a shuffle affects all PEs, and that any exchange signal affects both PE(i) and PE(Exchange(i)). Unlike Lawrie's omega network, no broadcast functions will be allowed. This network can produce all the permutations of the recirculating Shuffle-Exchange network. Thus, it has the advantage over the multistage Shuffle-Exchange network of being able to perform one to n-1 shuffles in one pass through the network.

Let $P = \{ (Si, Di) \mid 0 \le i < N \}$ be a permutation mapping of source PE address Si to destination PE address Di, where the binary representation of Si is $p_{n-1} \ldots p_1 p_0$ and that of Di is $d_{n-1} \ldots d_1 d_0$. A network passes a permutation P if and only if P conforms to the acceptable form for that network, and no conflicts result in the passage through the network [LAW75]. $Z \uparrow P$ means the network Z passes a permutation P.

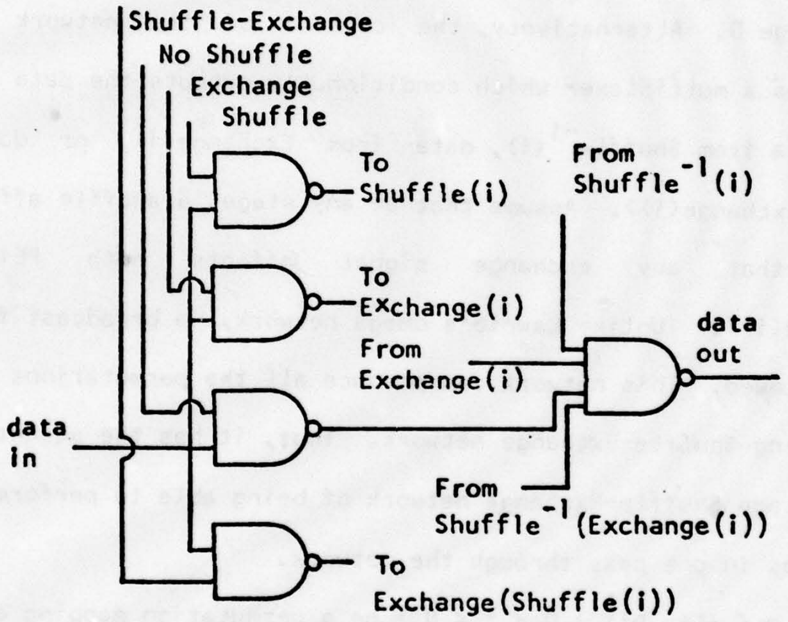


Figure IV.8: A Shuffle-No Shuffle-Exchange circuit for row i, $0 \leq i < N$.

__Theorem__ __IV.1__: An n-stage SNSE network can form the following permutations. Consider two pairs of acceptable source-destination tags pairs, (Si, Di) and (Sj, Dj). Then for all i and j, $0 \leq i, j < N$, the acceptable source destination pairs have the following form. For all t such that $0 \leq t < n$,

$$1)(A)\frac{Si}{2} \text{ modulo } 2^{n-(t+1)} = \frac{Di}{2^{t+1}} \text{ modulo } 2^{n-(t+1)}$$

$$(B) \text{AND } (Si \neq Sj \Rightarrow (\bigwedge_{b=0}^{t} (|\frac{Si}{2}| \text{ modulo } 2^{n-b-1} \neq |\frac{Sj}{2}| \text{ modulo } 2^{n-b-1} \quad \text{OR}$$

$$|\frac{Di}{2^{t-k}}| \text{ modulo } 2^{k+1} \neq |\frac{Dj}{2^{t-k}}| \text{ modulo } 2^{k+1}))),$$

Or, the pairs are of the form,

$$2) Si \neq Sj \Rightarrow (Si \text{ modulo } 2^{k} \neq Sj \text{ mod } 2^{k}$$

$$\text{OR } |\frac{Di}{2^{k}}| \text{ modulo } 2^{n-k} \neq |\frac{Dj}{2^{k}}| \text{ modulo } 2^{n-k})$$

for $1 \leq k \leq n$, where

$$\bigwedge_{b=0}^{t} (A(k))$$

is the logical AND of all A(k) for $0 \leq k \leq t$.

__Proof:__ An n-stage SNSE network, designated Z, will be analyzed by parts. Z_0 will refer to a network constructed with only an exchange function, i.e., $Z_0 = E$. Z_1 prefixes an Exchange-Shuffle stage to Z_0, i.e., $Z_1 = (ES)Z_0 = ESE$. Z_2 prefixes an Exchange-Shuffle to Z_1, i.e., $Z_2 = (ES)Z_1$

= (ES)(ES)Z_0 = (ES)(ES)E. For $0 < t < n$, $Z_t = (ES)Z_{t-1} = (ES)^tE$. The last prefix creates Z_n, an n stage Shuffle-Exchange network, by prefixing a shuffle stage to Z_{n-1}, i.e., $Z_n = S(ES)^{n-1}E = (SE)^n$. The permutations that Z can pass are the union of the permutations that Z_0, Z_1, . . . , and Z_n can pass.

Note that certain cases are not explicitly considered here. For example, a shuffle followed by k No Shuffles is equivalent to k No Shuffles followed by a shuffle. Also, a shuffle followed by two exchanges has the same effect as just a shuffle. The first cases are not considered in this argument. The second equivalent ones are. Without loss of generality, it is assumed that if stage i is a Shuffle, then so is stage j, for all j, $i > j \geq 0$.

Z_0 accepts the permutations

$$p_{n-1}\cdots p_1p_0 \dashrightarrow p_{n-1}\cdots p_1\overline{p}_0.$$

A conflict, C0, between two different sources Si and Sj results if and only if Di = Dj, that is

$$C0 = (|Si/2| = |Sj/2|) \text{ AND } (Di \text{ modulo } 2 = Dj \text{ modulo } 2).$$

So,

$$Z_0 \uparrow P \iff (|Si/2| = |Di/2|) \text{ AND}$$
$$(Si \neq Sj \Rightarrow \text{NOT}(C0)),\ 0 \leq i,j < N.$$

Recall that $Z_t = (ES)^tE$. For $0 < t < n$, the transition of data is

$$\begin{aligned} p_{n-1}\cdots p_0 &\dashrightarrow p_{n-1}\cdots p_1d_t \\ &\dashrightarrow p_{n-2}\cdots p_1d_td_{t-1} \\ &\dashrightarrow p_{n-3}\cdots p_1d_td_{t-1}d_{t-2} \\ &\cdots \\ &\dashrightarrow p_{n-1-t}\cdots p_1d_td_{t-1}\cdots d_0. \end{aligned}$$

The acceptable permutations are, thus,

$$|Si/2| \text{ modulo } 2^{n-(t+1)} = |Di/2^{t+1}| \text{ modulo } 2^{n-(t+1)}.$$

For $Si \neq Sj$, a conflict results after k shuffles and k+1 possible exchanges, $0 \leq k \leq t$, if and only if

$$Ct(k) = (|Si/2| \text{ modulo } 2^{n-k-1} = |Sj/2| \text{ modulo } 2^{n-k-1})$$
$$\text{AND } (|Di/2^{t-k}| \text{ modulo } 2^{k+1} = |Dj/2^{t-k}| \text{ modulo } 2^{k+1}).$$

Thus, the expression for a conflict, Ct, is

$$Ct= \bigvee_{k=0}^{t} (Ct(k)).$$

Therefore,

$$Z_t \uparrow P \Longleftrightarrow$$
$$(|Si/2| \text{ modulo } 2^{n-(t+1)} = |Di/2^{t+1}| \text{ modulo } 2^{n-(t+1)})$$
$$\text{AND } (Si \neq Sj \Rightarrow \text{NOT}(Ct)), \; 0 \leq i,j < N.$$

The combination of the permutations which Z_0 through Z_{n-1} pass is equivalent to the above statements 1) (A) and (B) in the statement of Theorem IV.1.

Z_n is a shuffle stage concatenated with Z_{n-1} and is an n-stage Shuffle-Exchange network. From [LAW75], it is seen that Z_n accepts a permutation P if and only if

$$Si \neq Sj \Rightarrow$$
$$\text{NOT}(Si \text{ modulo } 2^{k} = Sj \text{ modulo } 2^{k}$$
$$\text{AND } |Di/2^{k}| \text{ modulo } 2^{n-k} = |Dj/2^{k}| \text{ modulo } 2^{n-k}),$$
$$1 \leq k \leq n.$$

The expression is equivalent to statement 2) in the statement above of Theorem IV.1.

The n-stage network Z passes all permutations that are passed by $Z_0, Z_1, \ldots, Z_n$. []

The SNSE network can form all the permutations of the recirculating Shuffle-Exchange. If x passes through the recirculating network are needed, then $\lceil x/n \rceil$ passes through the SNSE network accomplish the same data transfer. This speed-up has been accomplished at the cost of a few extra gates per PE, 5*N*n for the SNSE network versus 7/2 N*n for the multistage Shuffle-Exchange, and for more control signals, n*N/2 for the multistage Shuffle-Exchange and (1(Shuffle) + N/2(Exchange) + N/2(Shuffle-Exchange)) * n signals for the SNSE network.

IV.4. Combinational Logic Multistage Networks and Pipelined Multistage Networks

In section 2, networks were analyzed as if they were one bit wide. Data words may be sent through the network bit serially, but other methods may be more efficient.

Let the width of a data word be W bits. A network may be designed as W planes, where each plane is a one bit wide network [LAW75]. As an example, consider the 10-stage generalized cube network for N = 1024 that was described in section 2. The number of packages required for W = 32 is 5974 * 32 = 191,168 integrated circuit packages, or about 187 per PE.

The amount of hardware could be reduced by a compromise. The data word could be divided into W/B = S segments of data, the network constructed as a B bit wide network, and then the data word passed in S passes through the network. In this manner, if the delay of the network is D, then the delay to pass the entire data word through the network is S*D. If the number of gates in a W wide network is G, the number of gates in the reduced network is G*B/W = G/S.

In order to show how this division of the network might be done, three sample hardware designs are presented. Design 1 moves data into and retrieves data from the network B bits at a time under software control. This method is slow, but requires no extra hardware for control. Design 2 multiplexes the S segments into and out of the network. The W bit data word is loaded into DTRin. An S-to-1 multiplexer supplies the network with each segment of data at the proper time. An S-to-1 demultiplexer retrieves the segments from the network and arranges them in DTRout. Since the maximum delay of the network is much less than one instruction cycle, this design is faster than design 1. But, it requires more hardware for the W bit wide DTRin and DTRout registers and the B bit wide multiplexer and demultiplexer. Design 3 constructs DTRin and DTRout from B S-bit shift registers. DTRin is made from parallel-in-serial-out registers, and DTRout is made from serial-in-parallel-out registers. Let $d_{W-1}...d_1d_0$ be a data word. The first register of DTRin stores bits $d_{S-1}...d_1d_0$. The second register stores bits $d_{2S-1}...d_{S+1}d_S$, and the last register stores bits $d_{W-1}...d_{W-S+1}d_{W-S}$. Each clock period, the least significant bit of each of the B shift registers of DTRin is presented to the network, the DTRin registers are shifted, and the next B bits are ready to be presented. During the clock period, these B bits propagate through the combinational logic of the network. Each clock period, at the output of the network, each of the B shift registers of DTRout receive one bit from the network. After S clock periods, the S bits in each of the B shift registers are presented as a W bit word to the PE. DTRin and DTRout will be treated as W-bit registers by the balance of the system. This design is faster than design 1 and requires less hardware than design 2.

For an extra cost, overlap parallelism may be added to a multistage network to reduce the total time to move S segments of data. Assume that the network is B bits wide, that DTRin and DTRout are W bits wide, and that costs for interfacing between the DTRs and the network are small compared to the cost of the network. Let the network be an n stage pipeline with registers of delay dr between each stage, each stage having delay dms. Figure IV.9 illustrates this arrangement.

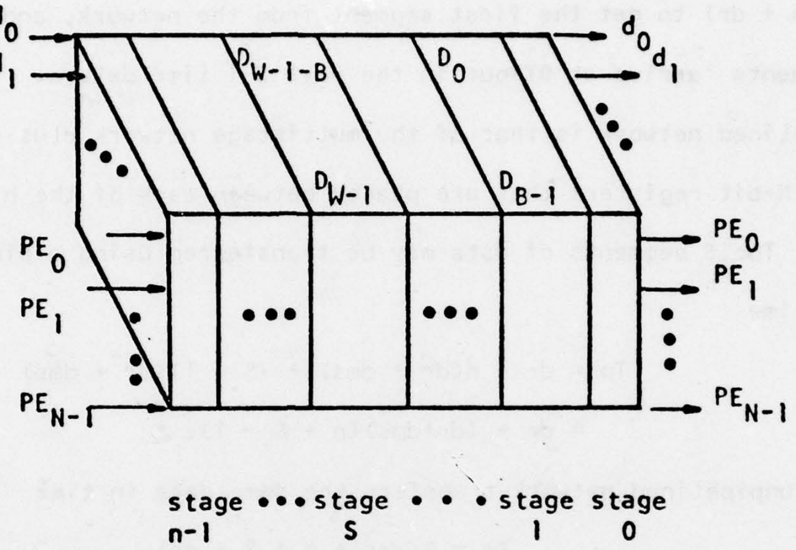


Figure IV.9: A model for a pipelined multistage network of width B for N PEs and data word $D_{W-1} \ldots D_1 D_0$.

The delay for the combinational logic multistage network is the time to load DTRin, plus the time to pass through the network, plus the time to load DTRout. The cost, n*cms*B, considers only the network and not DTRin and DTRout. The delay of the pipelined network is dr + n * (dms + dr) to get the first segment from the network, and the remaining segments arrive at DTRout in the next S-1 time delays. The cost of the pipelined network is that of the multistage network plus cr*(n-1)*B for the N-bit registers that are placed between each of the n stages.

The S segments of data may be transferred using a pipelined network in time

$$\begin{aligned} Tp &= dr + n(dr + dms) + (S - 1)(dr + dms) \\ &= dr + (dr+dms)(n + S - 1). \end{aligned}$$

The unpipelined network transfers the same data in time

$$Tm = S(dms * n + 2 * dr).$$

If the last segment is loaded into DTRout as the next segment is loaded into DTRin, then

$$Tm = dr + S(dms*n +dr).$$

Either formula can be used for the analysis, and the difference in the analysis is negligible.

Figures IV.10, IV.11, and IV.12 plot Tp vs Tm for various values of N and S, for dr = 9 ns and dms = 10 ns. These time approximations are based on Schottky logic according to [TI76]. In all three Figures, for S > 2, the pipelined multistage network passes data in less time than the combinational logic multistage network. As S grows, this time difference becomes more pronounced.

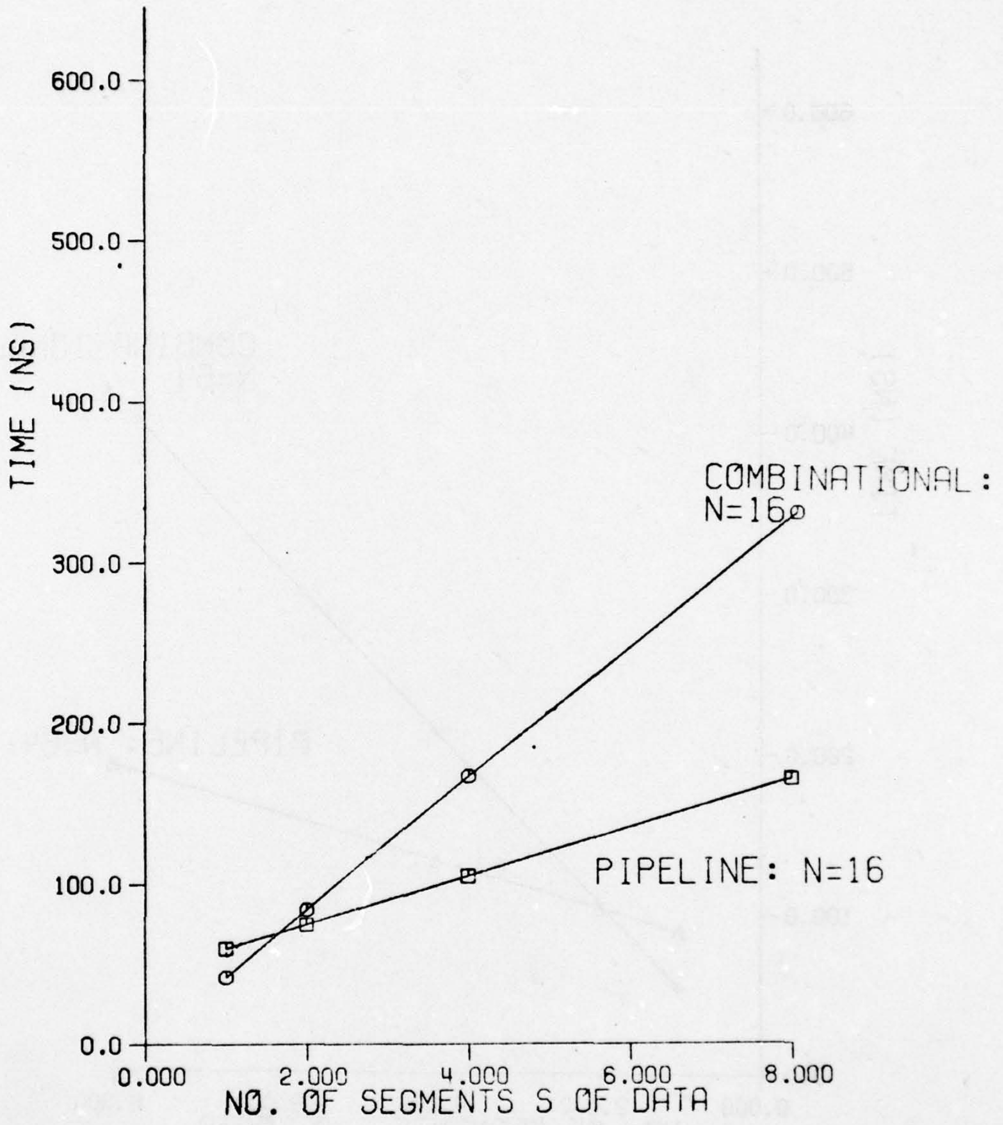


Figure IV.10: Tp vs. Tn for N = 16.

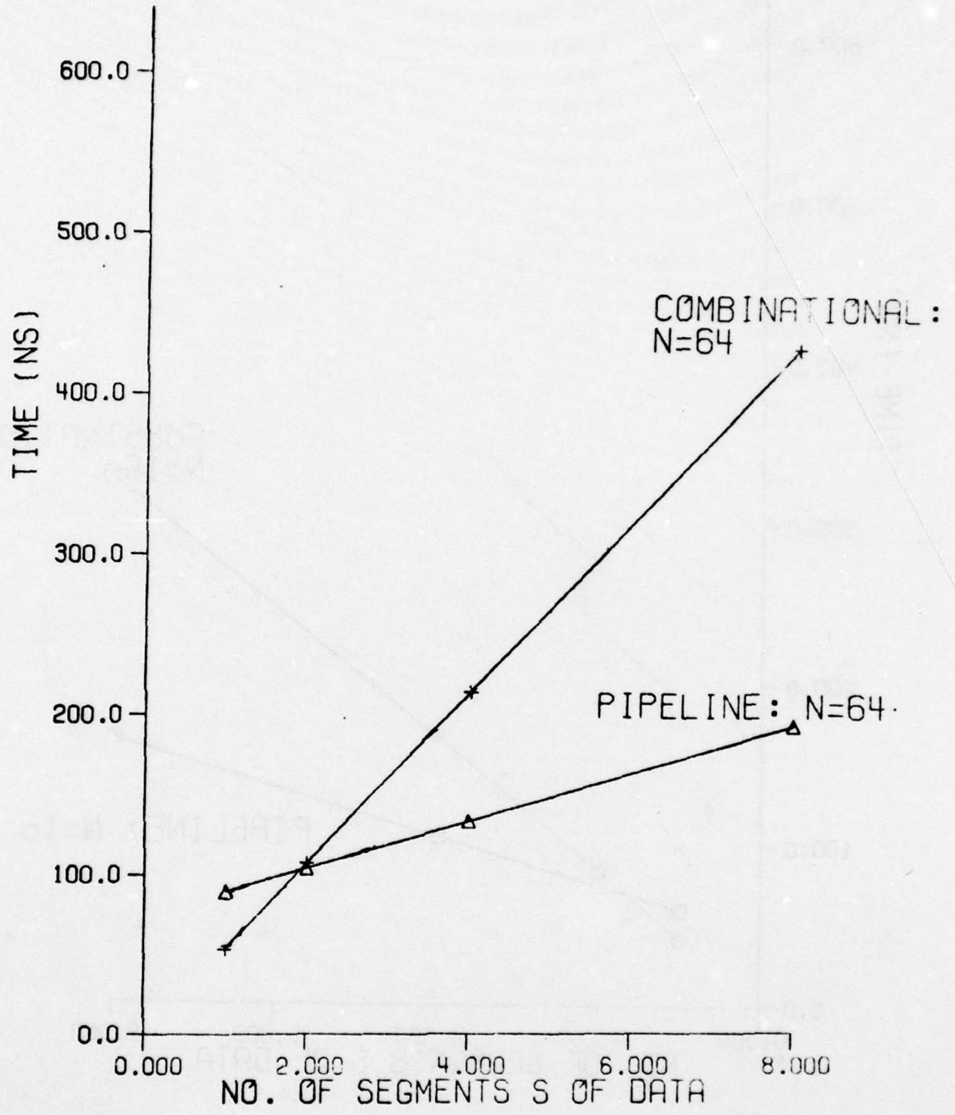


Figure IV.11: Tp vs Tm for N = 64.

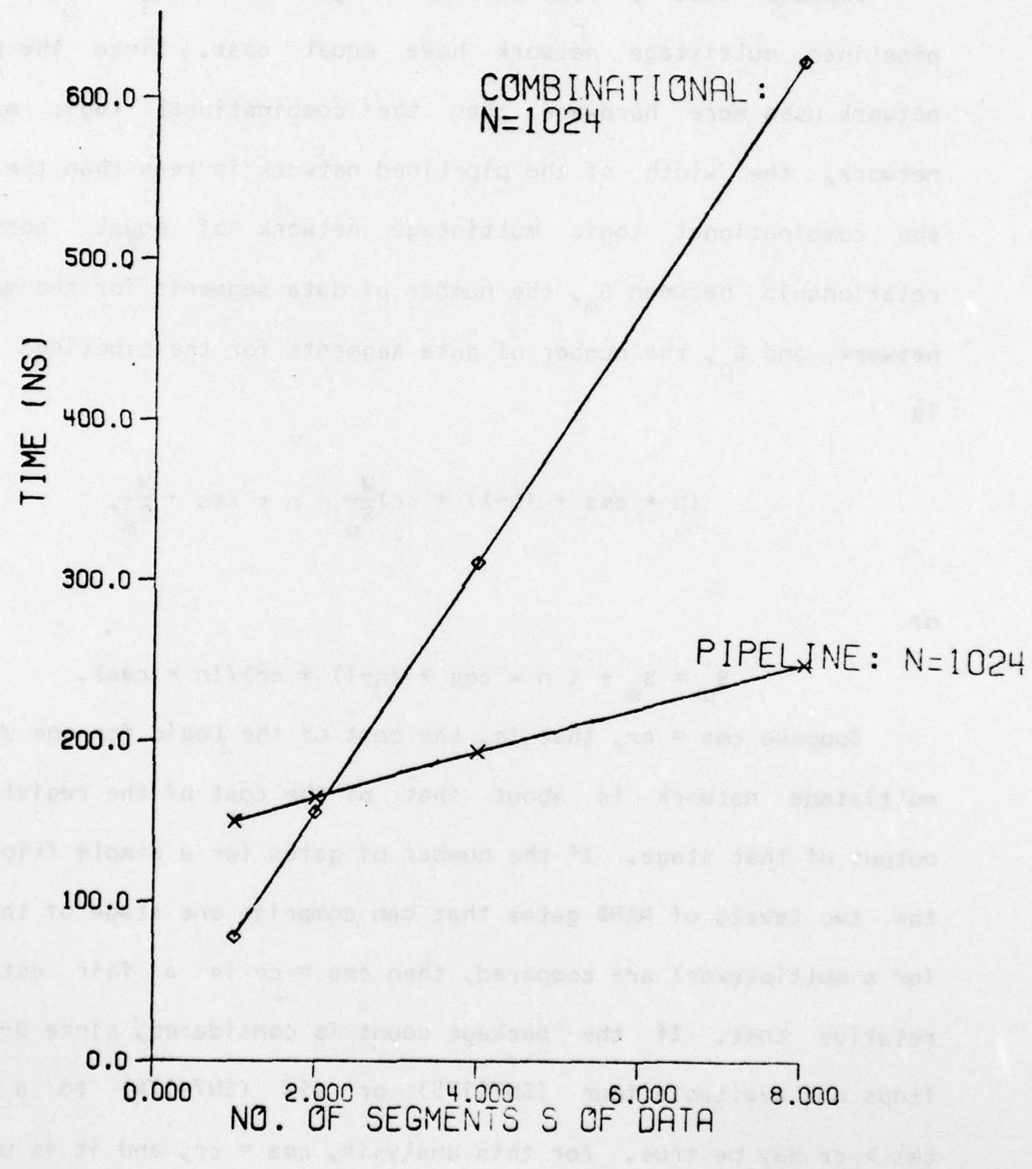


Figure IV.12: Tp vs Tm for N = 1024.

IV.5. Equal Cost Combinational Logic and Pipelined Multistage Networks

Suppose that a combinational logic multistage network and a pipelined multistage network have equal cost. Since the pipelined network uses more hardware than the combinational logic multistage network, the width of the pipelined network is less than the width of the combinational logic multistage network of equal cost. The relationship between S_m, the number of data segments for the multistage network, and S_p, the number of data segments for the pipelined network, is

$$(n * cms + (n-1) * cr)\frac{W}{S_p} = n * cms * \frac{W}{S_m},$$

or

$$S_p = S_m * (n * cms + (n-1) * cr)/(n * cms).$$

Suppose cms = cr, that is, the cost of the logic for one stage of a multistage network is about that of the cost of the register at the output of that stage. If the number of gates for a simple flip-flop and the two levels of NAND gates that can comprise one stage of the network (or a multiplexer) are compared, then cms = cr is a fair estimate of relative cost. If the package count is considered, since D-type flip flops are available four (SN74175) or six (SN74174) to a package, cms > cr may be true. For this analysis, cms = cr, and it is understood that this is a worst case estimate for pipelined network costs.

If cms = cr, then

$$(n * C + (n-1) * C)/S_p = n * C/S_m, \text{ or}$$

$$(n + (n-1)) * S_m = n * C * S_p, \text{ or}$$

$$S_p = (2*n - 1) S_m / n .$$

Table IV.2 lists S_p, S_m, Tp, and Tm for various values of N for combinational and pipelined networks of equal cost, assuming dms = 10 ns and dr = 9 ns. Figures IV.13, IV.14, and IV.15 illustrate the data transfer delay time variations for two equal cost networks. In order to move W bits through a network, for a small number of segments of data, the unpipelined network completes the data transfer in less time than the pipelined multistage network. However, as the number of segments grows, the pipelined multistage network requires less time to complete the transfer than the same cost combinational logic multistage network.

Table IV.2: N, S_p, S_m, Tm, and Tp for equal cost networks.

N	S_m	Tm	$\lvert S_p \rvert$	Tp
16	1	58	2	104
16	4	205	7	199
16	8	401	14	332
16	16	793	28	598
128	1	88	2	161
128	4	325	8	275
128	8	641	15	408
128	16	1273	30	693
1024	1	118	2	218
1024	4	445	8	332
1024	8	881	16	484
1024	16	1753	31	769

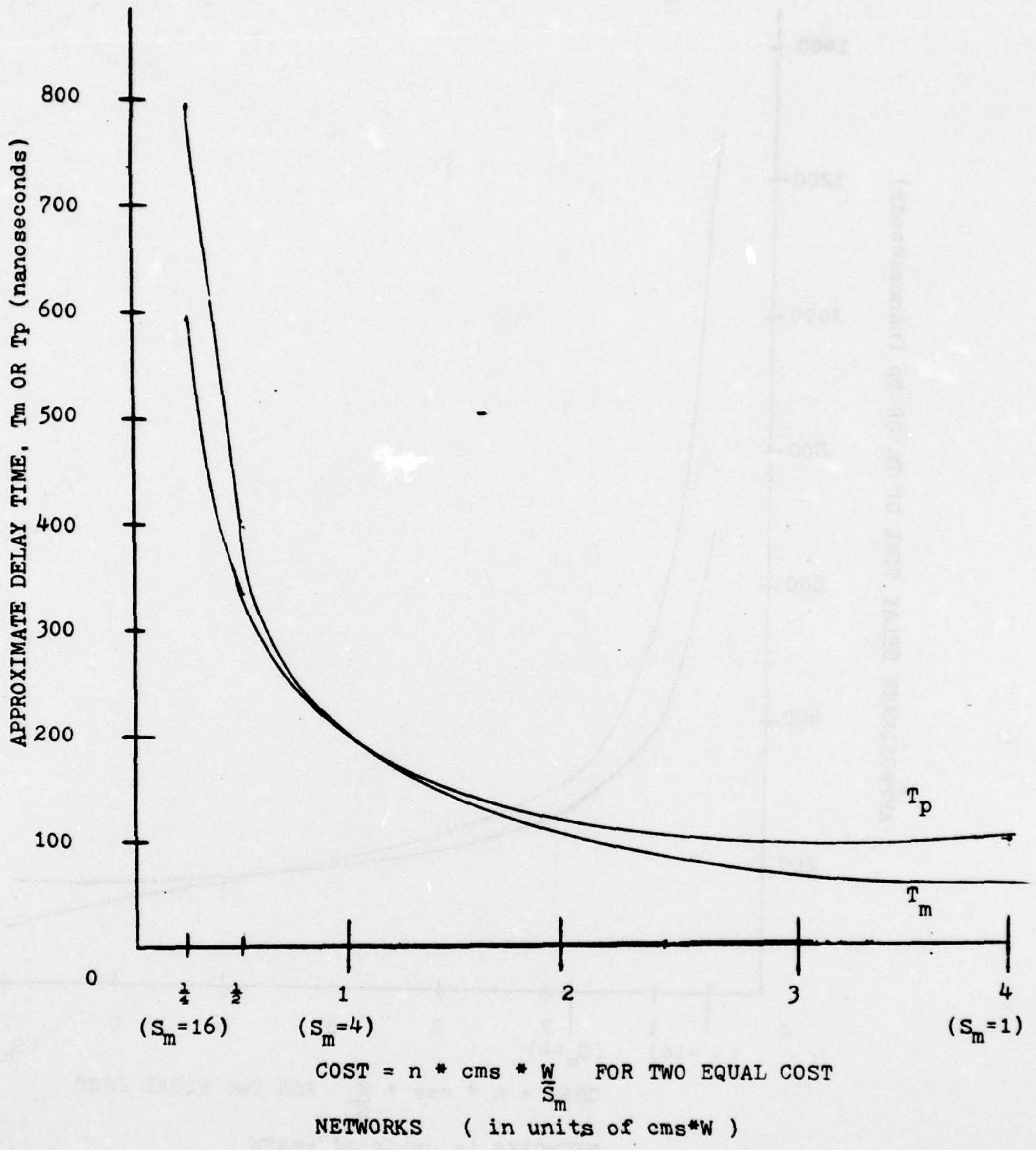


Figure IV.13: Cost vs delay for equal cost pipelined and combinational logic networks for N = 16.

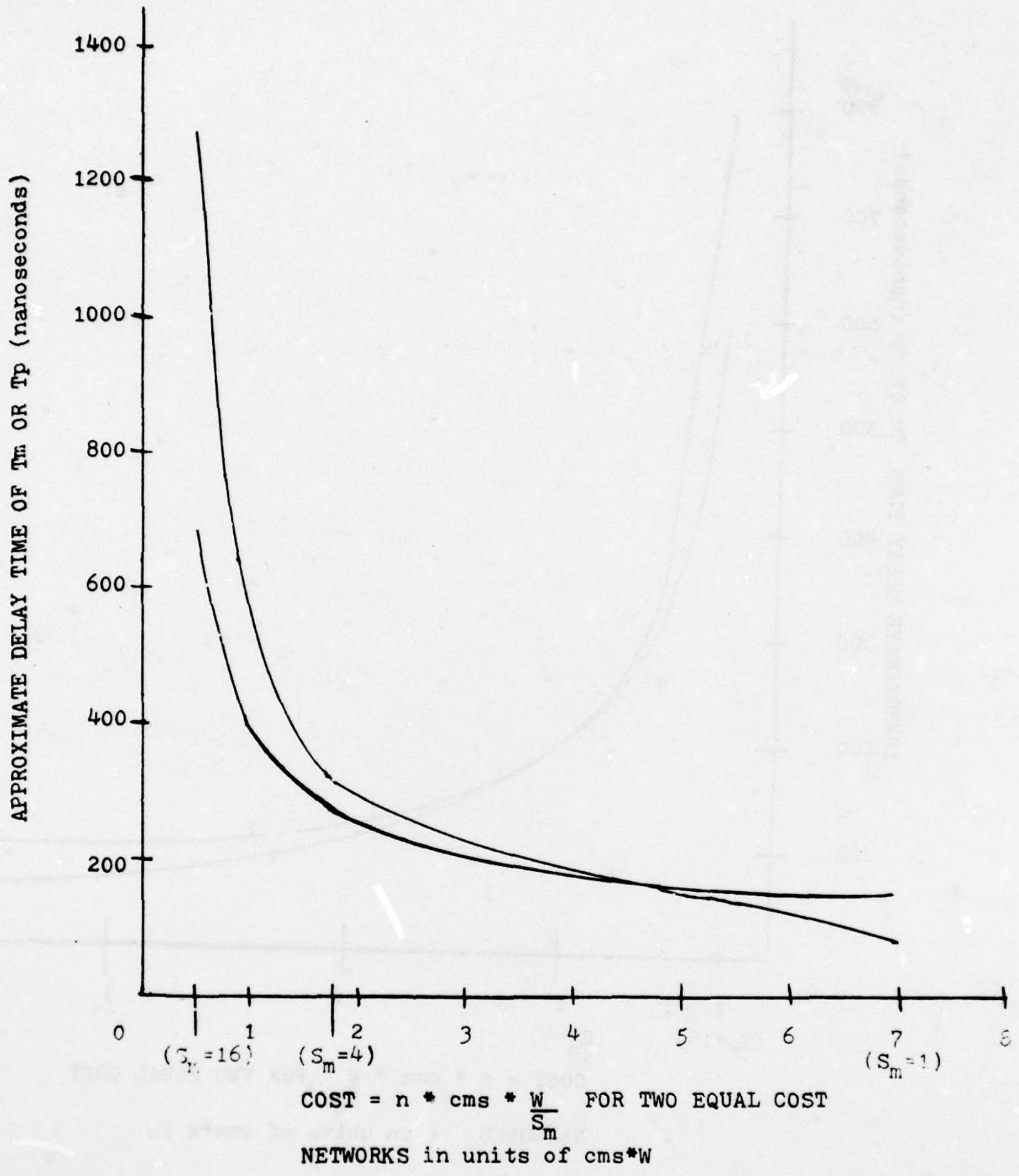


Figure IV.14: Cost vs delay for equal cost pipelined and combinational logic networks for N = 128.

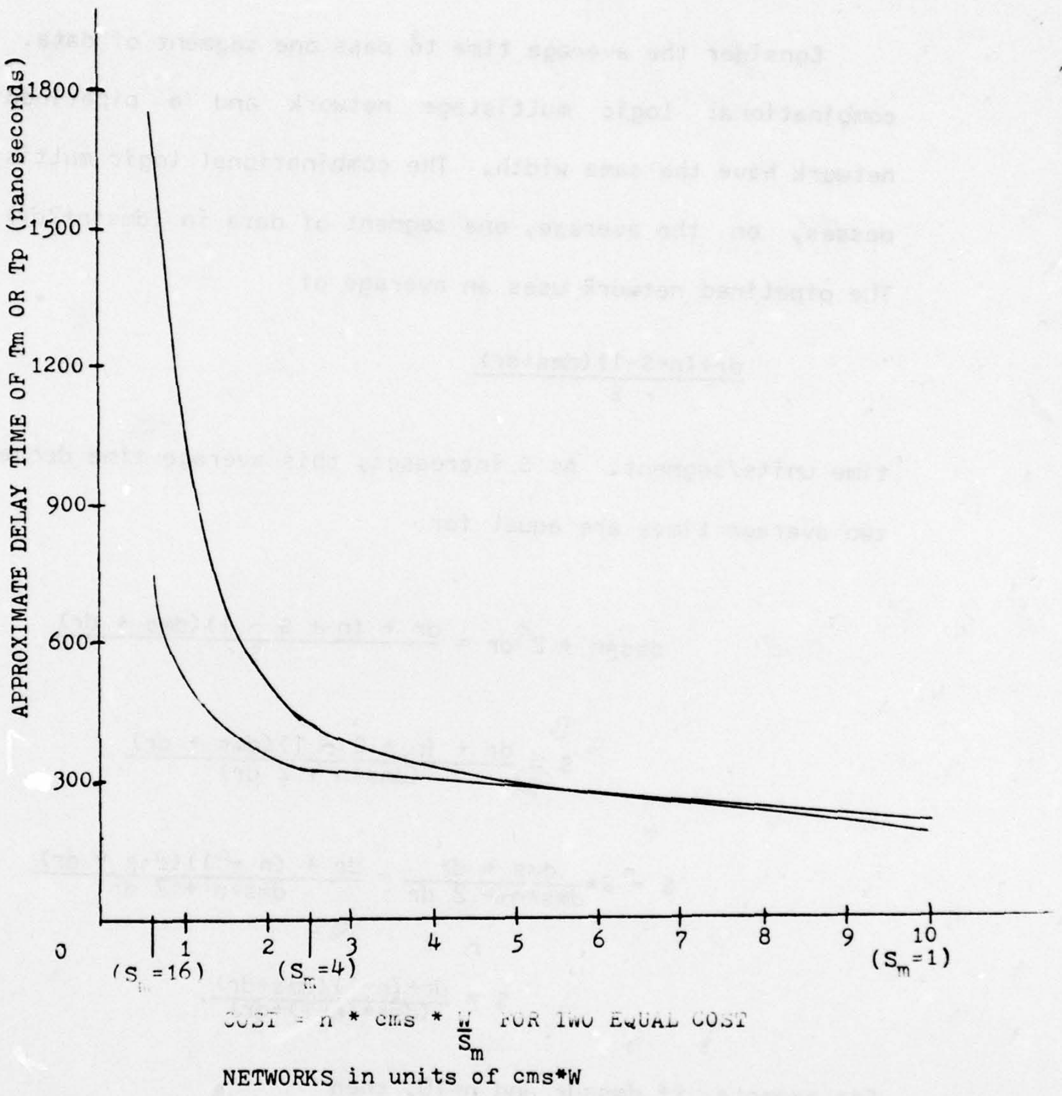


Figure IV.15: Cost vs delay for equal cost pipelined and combinational logic networks for N = 1024.

IV.6. Average Data Transfer Times

Consider the average time to pass one segment of data. Let both a combinational logic multistage network and a pipelined multistage network have the same width. The combinational logic multistage network passes, on the average, one segment of data in (dms*n+2dr) time units. The pipelined network uses an average of

$$\frac{dr+(n+S-1)(dms+dr)}{S}$$

time units/segment. As S increases, this average time decreases. These two average times are equal for

$$dms*n + 2\ dr = \frac{dr + (n + S - 1)(dms + dr)}{S}$$

$$S = \frac{dr + (n + S - 1)(dms + dr)}{(dms*n + 2\ dr)}$$

$$S - S*\frac{dms + dr}{dms*n + 2\ dr} = \frac{dr + (n - 1)(dms + dr)}{dms*n + 2\ dr}$$

$$S = \frac{dr+(n-1)(dms+dr)}{(dms*(n-1)+dr)}.$$

For example, if dms=dr and n=10, then

$$S = \frac{dr+(10-1)(2dms)}{(dms*(10-1+1))} = \frac{19}{10}.$$

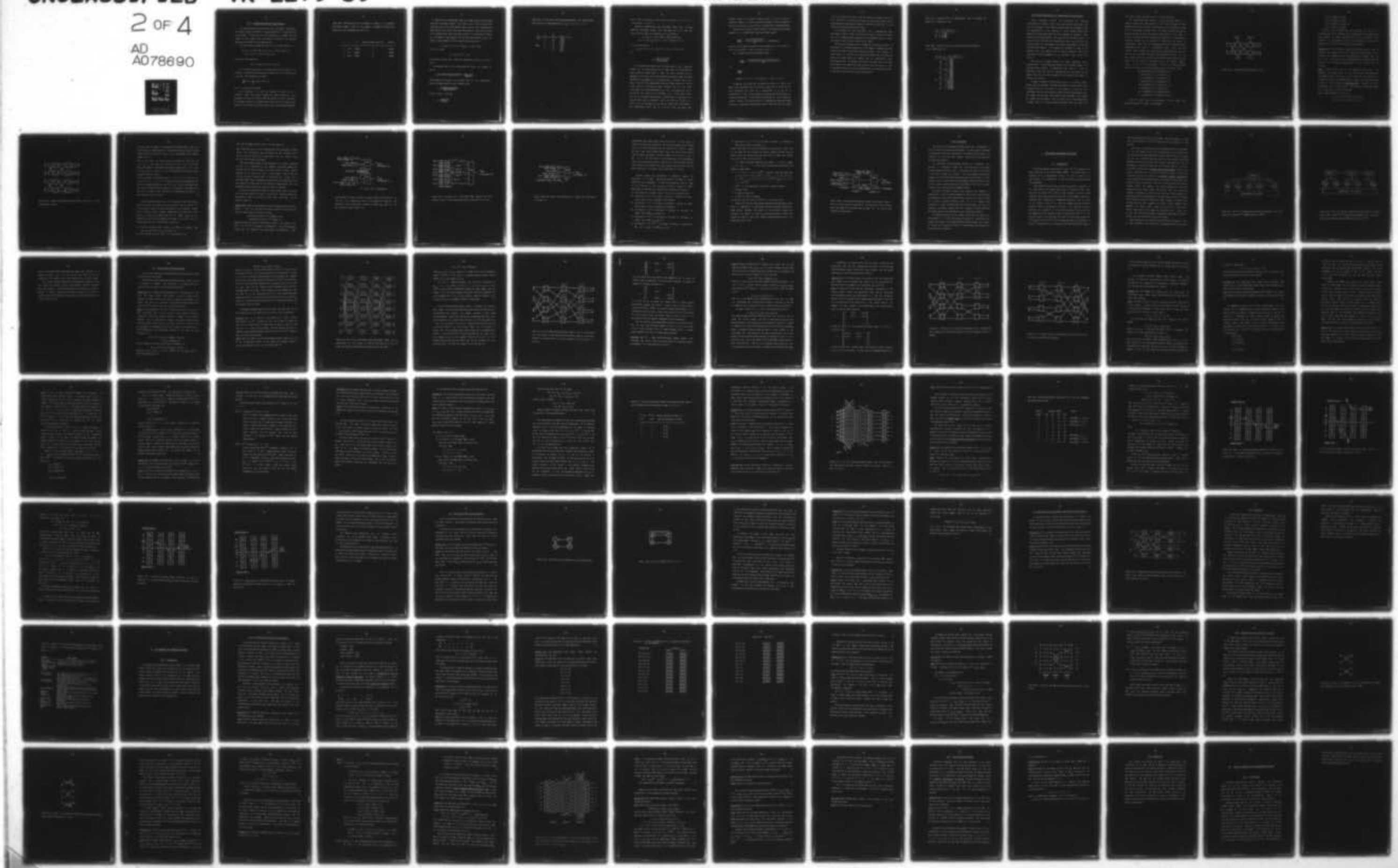
So, for $S \geq 2$, the average delay per segment is less for the pipelined network than for the combinational logic multistage network, although the total time to pass one data item may be greater, due to the time to fill and empty the pipe. This suggests that a pipelined multistage network is most applicable where many segments of data are passed, such as passing blocks of data at once rather than one data word.

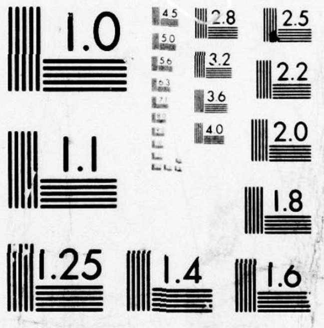
AD-A078 690 PURDUE UNIV LAFAYETTE IN SCHOOL OF ELECTRICAL ENGINEERING F/G 9/2
DESIGN AND ANALYSIS OF INTERCONNECTION NETWORKS FOR PARTITIONAB--ETC(U)
AUG 79 S D SMITH , H J SIEGEL AFOSR-78-3581

UNCLASSIFIED TR-EE79-39 AFOSR-TR-79-1272 NL

2 OF 4

AD
A078690




1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

IV.7. k-stage Pipelined Multistage Networks

An n stage pipeline may not be best for a particular application. The network can be divided into a k-stage pipeline, $0 < k \leq n$, where n/k is an integer. In this case, k-1 extra registers are added, and n/k stages of the network form one stage of the pipeline. The analysis above can be extended to a k-stage pipeline.

The time to pass S segments of data for the k-stage pipeline is

$$Tk = dr + k (n/k * dms + dr) + (S - 1)(n/k * dms + dr)$$
$$= dr + (k + S - 1)(n/k * dms + dr).$$

The cost of this network is

$$Ck = n * cms * B + (k-1) * cr * B.$$

If S is fixed, the value of k for which the delay is minimum is of interest. By taking the derivative with respect to k of Tk, and setting it to zero, this minimum can be found.

$$\frac{dTk}{dk} = dr + \frac{-n}{k^2} * dms * (S-1) = 0$$

or, k = -/(n * dms * (S-1)/dr).

If n = 10 and dms = dr, then the minimum Tk occurs for k = -/10(S-1). Table IV.3 shows Tk and k for various values of S. As S increases, the value of k which yields the minimum Tk also increases. The minimum value of Tk increases with k and S, but for a fixed S, the value of k in the table yields the minimum Tk for any pipelined network.

Table IV.3: Minimum value of k, the number of stages in a pipelined multistage network, given S, the number of segments of data to be passed, and the corresponding value of Tk.

S	k	Tk	nearest integer value n/k	Tk(n/k)
2	3.16	17*dms	2	19*dms
4	5.5	24*dms	5	25*dms
8	8.36	33*dms	10	35*dms

Suppose that a combinational logic multistage network and a k-stage pipelined multistage network have equal cost. Since a one bit wide pipelined network uses more hardware than a one bit wide combinational logic network, the width of the pipelined network is less than the width of the combinational logic multistage network. The relationship between S_m, the number of data segments for the multistage network, and S_k, the number of data segments for the k-stage pipelined network is

$$(n * cms + (k-1) * cr) * W/S_k = n * cms * W/S_m .$$

If cms = cr, then

$$S_k = S_m * (n + k - 1)/n$$

for networks of equal cost. Table IV.4 compares S_k and S_m for various n and k.

The average time for the k-stage pipeline to pass one segment of data is

$$\frac{dr + (dms * n/k + dr)(k+S-1)}{S} \; \frac{\text{time units}}{\text{segment}}.$$

This average time is equal to the average time for the combinational logic multistage network to pass a segment when

$$S = \frac{dr+(dms*n/k+dr)(k-1)}{dms*(n-n/k)+dr}.$$

If dms = dr and n = 10, then

$$S = \frac{k^2+10k-10}{11k-10}.$$

Table IV.4: For two equal cost multistage networks, one combinational logic and one a k-stage pipeline, $S_k = S_m * (n + k - 1) / n$.

N	k	S_m	S_k
64	2	4	4.67
		8	9.33
	6	4	7.33
		8	14.67
1024	2	4	4.4
		8	8.8
	5	4	5.6
		8	11.2

For k = 10, S = 1.9, which is the result of section 6. For k = 5, S = 1.44, and for k = 2, S = 1.

Consider a combinational logic multistage network and a k-stage pipelined multistage network that have equal cost. Let D = dms = dr. Then, since for equal cost networks, $S_k = S_m$ (n+k-1)/n,

$$Tm = S_m (n * dms + 2*dr) = S_m *(n+2)*D, \text{ and}$$

$$Tk = dr + (k + S_m * (n + k - 1)/n - 1)*(n/k + 1) * D.$$

Tk > Tm if and only if

$$S_m (n + 2) < 1 + (k + S_m * (n + k - 1)/n - 1)(n/k + 1),$$

or

$$S_m < \frac{n^2k - n^2 + nk^2}{n^2k-n^2+n-k^2+k}.$$

This relationship implies that, for some values of S_m, a pipelined network can be designed whose cost is about that of the combinational logic network, but whose delay to pass one datum (divided into S_k segments) is less than that of the unpipelined network to pass the datum (divided into S_m segments). To illustrate this, consider n=10, S_m=1, W=16. Then, for the combinational logic multistage network, B = 16, Cm = 160 C, and Tm = 12 D, where D = dms = dr and C = cms = cr. For k=10, for an equal cost pipelined network, S_k = 1.9. If we choose S_k = 2, B = 8, then Ck = 152 C and Tk = dr + (11)(dms + dr) = 23 D. Here, then, Tk > Tm for two equal cost networks. Alternatively, if S_m=4 and B = 4, then Cm = 40 C, and Tm = 4(10+2) D = 48 D. For k=10, S_k = 4 (10 + 10 - 1)/10 = 7.6. If we choose S_k = 8, then Ck = 2(10 + 9)C = 38 C and Tk = D(1 + (10+8-1)(2)) = 35 D. So, Tk < Tm for these two equal cost networks. Using the relation defined above, if $S_m > 2.3$, then a pipelined network is a more cost effective design, since for about the same cost, the delay is less that for an unpipelined multistage network.

Consider the following "figures of merit" for evaluating multistage networks. For a combinational logic multistage network,

$$\frac{Tm*Cm}{D*C} = \frac{Wt*S_m*(n+2)D*W/S_m*n*C}{D*C} = Wt*W*n*(n+2),$$

where Wt is the number of words per data transfer, W is the width in bits of a data word, D = dms = dr, and C = cms = cr.

For a k-stage pipelined multistage network,

$$\frac{Tk*Ck}{D*C} = \frac{D(1+(k+S_k*Wt-1)(n/k+1))*W/S_k(n+k-1)C}{D*C}$$

or,

$$\frac{Tk*Ck}{(D*C)} =$$

$$Wt*W(n/k + 1)(n + k - 1) + W/S_k(n + k - 1)(n + n/k + k).$$

In general, for a given Wt, the smaller the figure of merit, the better the performance for a given cost. Note that Tm * Cm/(D * C) is not a function of S_m, since Tm is proportional to S_m and Cm is proportional to $1/S_m$. As S_m increases, the delay time Tm increases, but the cost Cm decreases. As Wt grows, the magnitude of this delay-cost product reflects the unsuitability of a combinational logic multistage network for transfering large blocks of data. Note also that Tk * S_k/(D

* C) is a function of S_k, but is not as sensitive to changes in Wt as is Tm * Cm/(D * C). As Wt and S_k increase, the delay-cost product for the pipelined multistage network changes to reflect the usefulness of the network for moving large blocks of data.

As an example, let n=10 and W=16. For a combinational logic multistage network, Table IV.5 shows the figures of merit for various Wt. For a k-stage pipelined network, the corresponding figures of merit are shown in Table IV.6 for various Wt, S_k, and k.

This analysis indicates that an unpipelined multistage design is preferable to a pipelined design only if the number of segments per data word and the number of words per transfer are small. For example, if large blocks of data are to be transferred between PEs, the pipelined network is a better choice for a design than the combinational logic multistage network. If, however, the width of the data word is the same as the width of the network and data transfers occur one word at a time at infrequent intervals, then the combinational logic multistage network is more cost effective than the pipelined network.

Table IV.5: Figures of merit for combinational logic multistage networks for N = 1024 and W = 16.

Wt	Tm * Cm/(D * C)
1	1920
8	15,360
16	30,720

Table IV.6: Figures of merit for pipelined multistage networks for N = 1024 and W = 16.

Wt	S_k	k	Tk * Ck/(D * C)
1	1	10	6992
	8		1406
	16		1007
	1	5	4480
	8		1148
	16		910
8	1	10	11,284
	8		5662
	16		5263
	1	5	9184
	8		5852
	16		5614
16	1	10	16,112
	8		10,526
	16		10,127
	1	5	14,560
	8		11,228
	16		10,990

IV.8. Custom Integrated Circuit Implementation Considerations

Custom integrated circuits are attractive for building interconnection networks since delay times and chip counts and, thus, cost of the network, can be reduced. Another potential advantage of such an implementation is the reduction of wires between stages of the network. The connections that such wires represent can become a major portion of the system expense. But, some restrictions must be made. In order to be cost effective, the implementation must use one chip as the building block for the network, and connect these modules to form the interconnection network. If the modules are packaged in dual inline packages, then the number of pins is limited to about 60. More pins, and so more flexible packaging, can be obtained using ceramic chip carriers.

This section will suggest designs for custom integrated circuit implementation of multistage Cube (Figure III.6) and PM2I (Figure III.7 and Figure IV.3) networks. It assumes that the logic on a chip is essentially free and that the guiding factors in any design are the number of pins per chip and the number of wires between chips needed to construct the network.

To reduce the number of wires that are not on a building module, several means are available. Consider the generalized cube network shown in Figure III.6. For any stage of the network, if more than one interchange box for that stage were placed on the building module, some wire length could be saved. More wires can be eliminated if two stages are merged onto one module. That is, given a set of interchange boxes in stages i and i-1, if the two stages are merged onto one chip, the wires out of stage i and into stage i-1 can be eliminated.

For a generalized cube network, if each building module is a complete generalized cube network of size 2^i, then the number of wires between chips can be minimized. One possibility is the generalized cube for $2^2 = 4$ PEs shown in Figure IV.16. This module requires four control signals, 8W pins for input and output of data, where W is the width in bits of the data word to be transfered, and two pins for power/ground. A further savings can be realized by noting that the module is combinational logic, and so the boolean functions it realizes can be implemented by two levels of combinational logic. Thus, the data transfer delay through the module can be reduced from four levels of logic to two. The four functions can be expressed as follows. Let the four inputs be I0, I1, I2, and I3, and the four outputs be O0, O1, O2, and O3. For two function interchange boxes, four control signals, c(i,j), $i, j \in \{0, 1\}$ are available. If c(i,j) = 1, then an exchange occurs. If c(i,j) = 0, then the straight state is selected. Then,

$$O0 = \overline{c}(0,0)\overline{c}(0,1)\ I0 + c(0,0)\overline{c}(0,1)\ I2$$
$$+ \overline{c}(1,0)c(0,1)\ I1 + c(1,0)c(0,1)\ I3$$
$$O1 = \overline{c}(1,0)\overline{c}(0,1)\ I1 + c(1,0)\overline{c}(0,1)\ I3$$
$$+ \overline{c}(0,0)c(0,1)\ I0 + c(0,0)c(0,1)\ I2$$
$$O2 = \overline{c}(0,0)\overline{c}(1,1)\ I2 + \overline{c}(1,0)c(1,1)\ I3$$
$$+ c(0,0)\overline{c}(1,1)\ I0 + c(1,0)c(1,1)\ I1$$
$$O3 = \overline{c}(1,0)\overline{c}(1,1)\ I3 + \overline{c}(0,0)c(1,1)\ I2$$
$$+ c(1,0)\overline{c}(1,1)\ I1 + c(0,0)c(1,1)\ I0.$$

For $N = 2^n$, stages i and i+1 can be merged. The four inputs into the building module at stage i have addresses

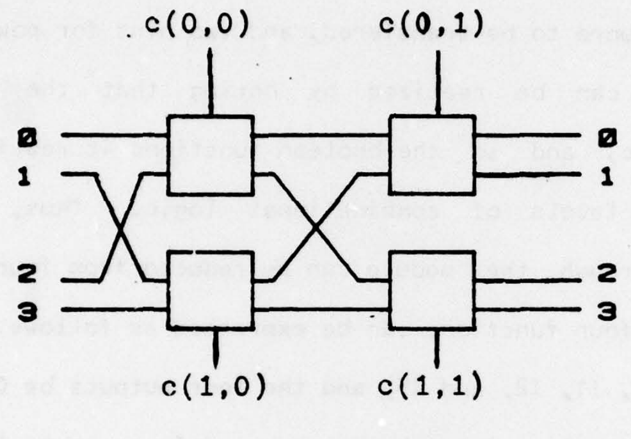


Figure IV.16: A generalized cube network for N = 4.

$p_{n-1} \cdots p_{i+1} 00 p_{i-2} \cdots p_1 p_0$,

$p_{n-1} \cdots p_{i+1} 01 p_{i-2} \cdots p_1 p_0$,

$p_{n-1} \cdots p_{i+1} 10 p_{i-2} \cdots p_1 p_0$, and

$p_{n-1} \cdots p_{i+1} 11 p_{i-2} \cdots p_1 p_0$.

There are other possibilities for selecting the logic for a building module for an interconnection network. But, for the generalized cube network, the module described by the four equations above represents an implementation which is least expensive in terms of the number of off chip wires required to build the network.

Theorem IV.2: Let the modules in Figure IV.16 be the chip, C1, used to implement two stages, i and i-1, of a generalized cube network. Let C2 be another chip composed of four interchange boxes in stages i and i-1, and C2 ≠ C1. Then, for any C2, an implementation of a generalized cube using C1 is a less expensive implementation of the network than one using C2.

Proof: In Figure IV.17, two C1 chips implement the circuit. One implements interchange boxes 0, 1, 2, and 3, and the other implements interchange boxes 4, 5, 6, and 7. No off chip wires are required for connections between interchange boxes in stage i and interchange boxes in stage i-1. Call these wires interstage wires. Consider the eight interchange boxes of Figure IV.17. Any chip C2 as defined above spans four of these boxes in one of three ways.

Case 1: Here, C2 groups four boxes as

(0, 1, 4, 5), and (2, 3, 6, 7), or

(0, 1, 6, 7), and (2, 3, 4, 5).

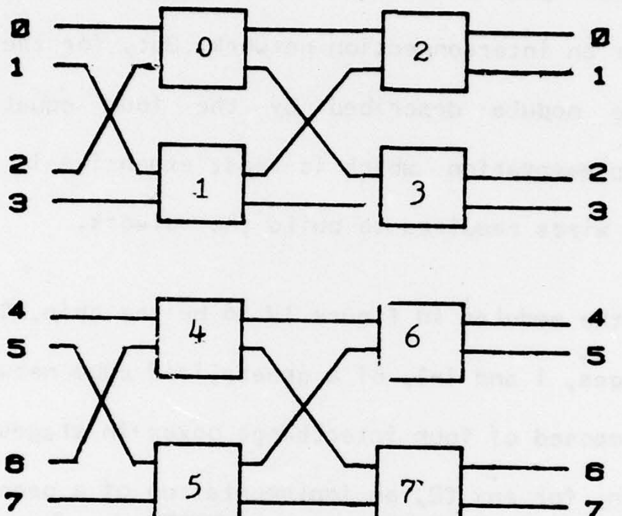


Figure IV.17: Eight interchange boxes from stages i and i-1 of a generalized cube network.

For this case, the number of interstage wires between stage i and i-1 is N. Therefore, an implementation of the generalized cube using C2 is more expensive than one using C1, which has no interstage wires between stages i and i-1.

Case 2: C2 spans the eight boxes by picking one from the set (0, 1, 2, 3) and three from the set (4, 5, 6, 7), or vice versa. Then, the number of interstage wires between stages i and i-1 is four for each pair of chips C2. So, any implementation using C2 is more expensive than one using C1.

Case 3: C2 groups interchange boxes (1, 3, 4, 6), (0, 2, 4, 6), (0, 2, 5, 7), or (1, 3, 5, 7). Because the four interchange boxes do not form a complete network of size $2^2 = 4$, as C1 does, interstage wires must connect different chips of type C2. The number of interstage wires is, for $N > 4$, $2(N/4) = N/2$. Therefore, any implementation using C2 is more expensive than one using C1. □

A multistage PM2I network can be constructed from multiplexers which select one of several inputs as the output from the circuit. For this discussion, assume that each switching cell of the network (Figure III.7) receives control signals independent of any other cell of the network. At stage i, an output line from the stage, P_{out}, for the switching cell position with address P at stage i, $n \geq i \geq 0$, $0 \leq P < N$, selects data from one of three inputs. These are

(1) with the "straight across" control, i.e., $PM2_{+i} = 0 = PM2_{-i}$, from P_{in}, the line which is P_{out} for stage i+1,

(2) with the $PM2_{-i}$ control, from $P + 2^i$ from stage i+1, or

(3) with the $PM2_{+i}$ control, from $P - 2^i$ from stage i+1.

Such a selection circuit can be represented by the multiplexer of Figure IV.18. This multiplexer uses 4W gates and two inverters for the controls, 3W data input lines, W data output lines, two control lines, and two lines for power and ground.

Multistage PM2I networks can be implemented as custom integrated circuits by a combination of two methods. One method combines the circuitry for two lines at stage i, as shown in Figure IV.19. Another combines the circuitry for line P from stages i and i+1 into one circuit, as shown in Figure IV.20. In specifying the circuits for a multistage PM2I network implementation, assume that all mappings of data from input PEs to output PEs are not necessarily permutations, that is, broadcasting data from one PE to several others is allowed.

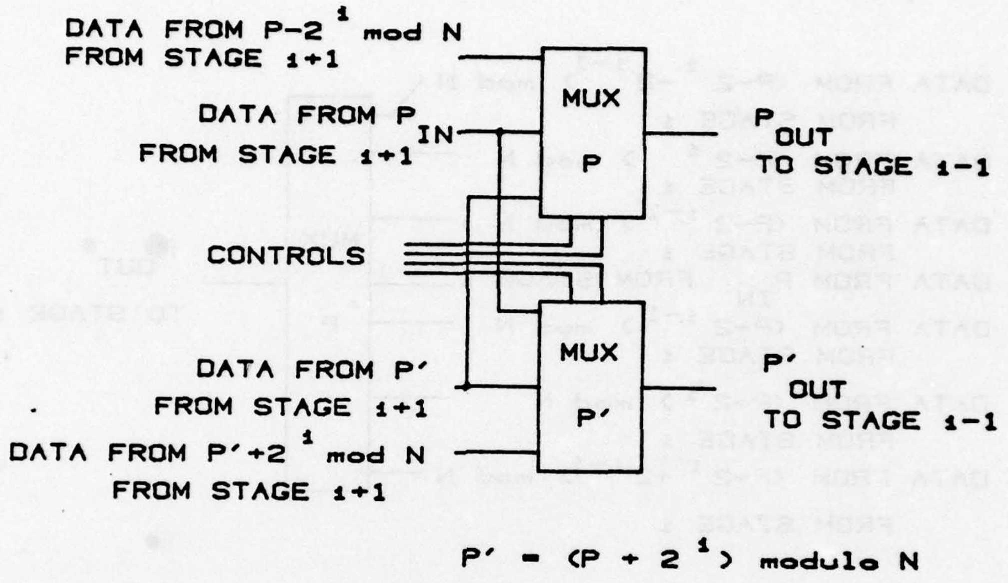
One way of decreasing the number of wires between chips is to merge two or more multiplexers at stage i into one chip. Consider merging two multiplexers, P and P', into one chip with input from stage i+1 and output to stage i-1.

<u>Theorem IV.3</u>: Suppose two multiplexers are to be merged into one chip at stage i. Then, the choices which remove the most off-chip wires and have the lowest number of pins per chip are

multiplexers P and $(P + 2^i)$ modulo N, and

multiplexers P and $(P - 2^i)$ modulo N.

<u>Proof</u>: Consider the choice of multiplexers P and $(P + 2^i)$ modulo N (the case of P and $(P - 2^i)$ modulo N is analogous). For this discussion, assume that all additions and subtractions are modulo N. Both



DATA FROM P-2 i mod N FROM STAGE i+1
DATA FROM P IN FROM STAGE i+1
MUX
P
P OUT TO STAGE i-1
CONTROLS
DATA FROM P′ FROM STAGE i+1
DATA FROM P′+2 i mod N FROM STAGE i+1
MUX
P′
P′ OUT TO STAGE i-1
P′ = (P + 2 i) modulo N

Figure IV.18: A circuit for line P at stage i selects data from line P, line $(P + 2^i)$ modulo N, or line $(P - 2^i)$ modulo N of stage i+1. The multiplexer selects some input to become the output P_{out} based on the set of control signals input to it.

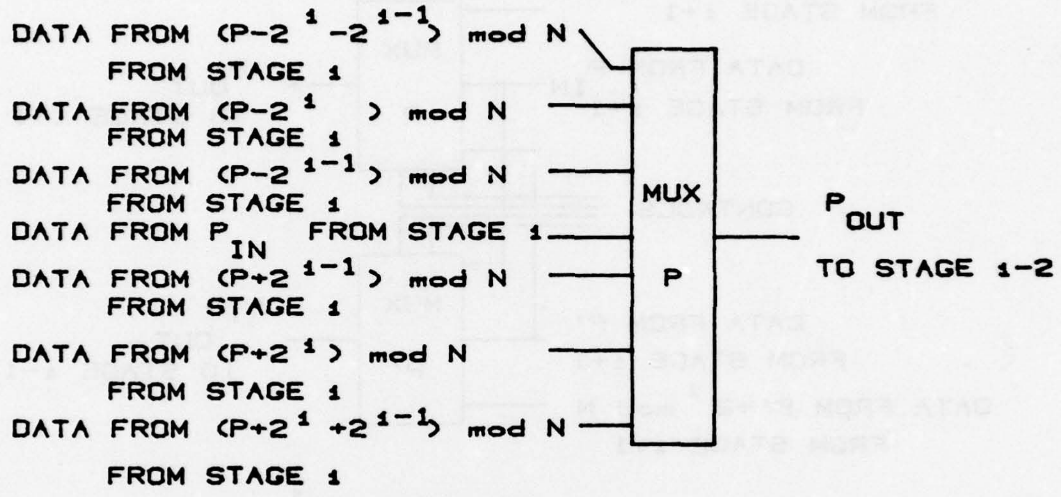

DATA FROM (P-2^i -2^(i-1)) mod N FROM STAGE i
DATA FROM (P-2^i) mod N FROM STAGE i
DATA FROM (P-2^(i-1)) mod N FROM STAGE i
DATA FROM P_IN FROM STAGE i
DATA FROM (P+2^(i-1)) mod N FROM STAGE i
DATA FROM (P+2^i) mod N FROM STAGE i
DATA FROM (P+2^i +2^(i-1)) mod N FROM STAGE i
MUX
P
P_OUT
TO STAGE i-2

Figure IV.19: At stage i of a multistage PM2I network, two multiplexers, P and P' can be constructed on one integrated circuit chip.

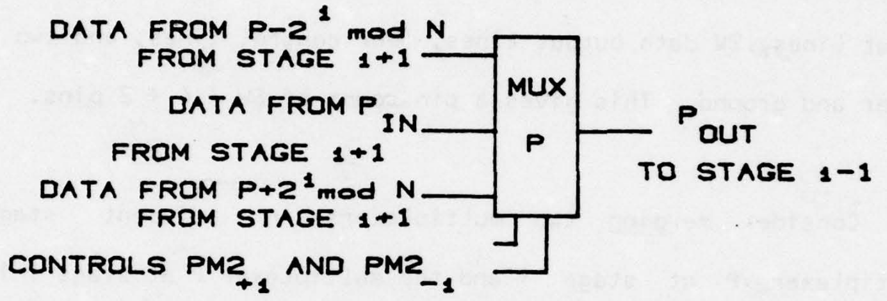


Figure IV.20: The circuits for multiplexer P at stage i and multiplexer P at stage i-1.

multiplexers have common inputs from lines P and P + 2^i from stage i+1 that do not need to be replicated. The chip then becomes that of Figure IV.19. This chip has 8W gates and four inverters for control signals, 4W data input lines, 2W data output lines, four control lines, and two lines for power and ground. The total number of pins for this chip is 6W + 4 + 2. Any other choice of two multiplexers with no common data input lines has 8W gates and four inverters for control signals, 6W data input lines, 2W data output lines, four control lines, and two lines for power and ground. This gives a pin count of 8W + 4 + 2 pins. []

Consider merging two multiplexers in adjacent stages, the multiplexer P at stage i and the multiplexer P at stage i-1. Figure IV.20 shows this arrangement. The data which become the input to stage i-2 can originate in the following positions at stage i+1. Again, assume that all additions and subtractions are modulo N arithmetic.

(1) multiplexer P, if stages i and i-1 are set to straight across,

(2) multiplexer P + 2^i, if at stage i the datum is affected by $PM2_{-i}$, and at stage i-1 by a straight across movement,

(3) multiplexer P - 2^i, if at stage i the datum is affected by $PM2_{+i}$ and at stage i-1 by a straight across movement,

(4) multiplexer P - 2^{i-1}, if the datum is affected by "straight" at stage i and by $PM2_{+i}$ at stage i-1,

(5) multiplexer P + 2^{i-1}, if the datum is affected by "straight" at stage i and by $PM2_{-i}$ at stage i-1,

(6) multiplexer P + 2^i + 2^{i-1}, if at stage i the datum is affected by $PM2_{-i}$ and at stage i-1 by $PM2_{-(i-1)}$, and

(7) multiplexer $P - 2^i - 2^{i-1}$, if at stage i the data is affected by $PM2_{+i}$ and at stage i+1 by $PM2_{-(i-1)}$.

Figure IV.21 shows the resulting multiplexer for position P. This chip has 8W gates and four inverters for control signals, 7W input lines, W output lines, four control lines, and two lines for power and ground, for a total of 8W + 6 pins per chip.

If s stages are merged beginning with stage i, $n > i \geq s-1$, stages i, i-1, i-2, . . . , i - (s-1), then the resulting chip on which the network is based needs

-- $(1 + 2 + 4 + \ldots + 2^s)\, W = (2^{s+1} - 1)W$ input lines, one from each of P, $P + 2^i$, $P - 2^i$, $P + 2^{i-1}$, $P - 2^{i-1}$, ..., $P + 2^i + 2^{i-1}$, $P - 2^i - 2^{i-1}$, ..., $P + 2^i + 2^{i-1} + \ldots + 2^{i-(s-1)}$, and $P - 2^i - 2^{i-1} - \ldots - 2^{i-(s-1)}$.

-- $(2^{s+1} - 1 + 1)$ W gates plus inverters for control signals,

-- W output lines,

-- 2s control lines, and

-- 2 lines for power and ground.

This gives a total pin count of $2^{s+1}W + 2s + 2$ pins per chip.

Theorem IV.3 and the stage merging techinque presented above can be using in combination to meet design specifications and limitations. Stage merging decreases the number of wires between stages, but increases the number of inputs to each building module. Theorem IV.3 reduces the number of input wires needed by combining multiplexers with common data input lines.

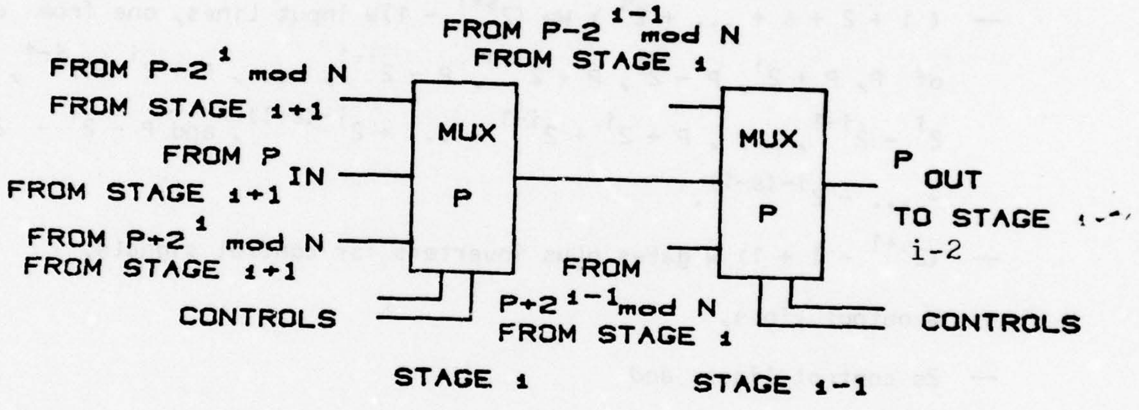


Figure IV.21: One multiplexer behaves the same as two stages, stages i and i-1, of the network for position P. The equivalent circuit receives input from stage i+1 and outputs data to stage i-2. It selects data from one of seven places.

IV.9. Conclusions

The Cube, Shuffle-Exchange, and PM2I networks were implemented as both recirculating and multistage networks. For each design, the amount of hardware and delay were calculated. The designs provide a basis for comparing the cost and data transfer time delays of multistage and recirculating networks.

The Shuffle-No Shuffle-Exchange network was introduced. This network is a compromise between the recirculating and multistage Shuffle-Exchange networks, since it can perform from one to n-1 shuffles in one pass through the network. For a small amount of additional hardware, the Shuffle-No Shuffle-Exchange network can perform all data transfers that the multistage Shuffle-Exchange network performs, as well as all the data transfers which can be performed in from 1 to n-1 passes through a recirculating Shuffle-Exchange network.

Pipelined multistage networks were examined and compared to combinational logic multistage networks of equal delay to pass data and to combinational logic multistage networks of equal cost. Both n-stage and k-stage pipelined networks, where n/k is an integer, were considered. Insight has been gained into the performance and cost of pipelined multistage networks and the trade-offs between pipelined multistage networks and combinational logic multistage networks.

The analyses developed in this chapter can be used to evaluate the effectiveness of a given interconnection network for a specified task. Chapter VII presents three image processing tasks and uses the results of Chapter IV to compare the times for interprocessor data transfer for several different networks.

V. PARTITIONING INTERCONNECTION NETWORKS

V.1. Introduction

Multimicroprocessor systems with as many as 2^{16} processors are being predicted for the future [PEA77, SUB77]. This implies that it is important for an interconnection network for such a system to be partitionable into subnetworks so that the system can be partitioned into subsystems.

Some computations may be more efficiently executed if the N PEs of the SIMD machine can be partitioned into 2^{n-r} groups of 2^r PEs. Each group may then behave like a smaller SIMD computer, and the system resources may be more efficiently utilized. As an example, consider an algorithm that is composed of k independent procedures, each of which can be executed using 2^r or fewer PEs. Suppose that each procedure requires at most j time periods to execute. The algorithm could be performed one procedure at a time, using at most 2^r PEs at any time. The computation would require at most j*k time periods, and $2^n - 2^r$ PEs would be unused. If on the other hand, the 2^n PEs were partitioned into 2^{n-r} groups of 2^r PEs, 2^{n-r} of the k procedures could be performed in parallel (assuming that all groups use the same instruction stream or that multiple control units are available). Now, the computer is more fully utilized, and if $k \leq 2^{n-r}$, the algorithm can be executed in j time periods.

Two classes of partitioning for an SIMD machine will be considered: (1) use the system as one to 2^{n-r} SIMD machines, each having 2^r PEs, each performing the same algorithm (using the same instruction stream), but each on a different data set; and (2) use the system as many SIMD machines of varying sizes, each size a power of two, where each may be performing a different algorithm on a different data set. The first class will be referred to as <u>single control partitioning</u>, since only one control mechanism will serve all of the PEs, as shown in Figure V.1. The second class will be referred to as <u>multiple control partitioning</u>, since a separate control mechanism will be required for each instruction stream, as shown in Figure V.2. Note that multiple control partitioning includes having, for example, four groups following one instruction stream, while four other groups each follow their own instruction streams. Furthermore, it may be the case that two or more groups may be executing different SIMD programs on the same logical data set, although each group would have its own physical copy. Examples of machines such as these that have been proposed are [NU77, SIE78a, SMSM78, BS78, BFHP79]. The single control class is a special case of the multiple control class, where all the groups use one control unit. Since there is only one instruction stream, a conventional SIMD machine, with one control unit, can provide this instruction stream.

When processors of the machine are partitioned into groups of size 2^r, each submachine must function as an independent entity. So, each

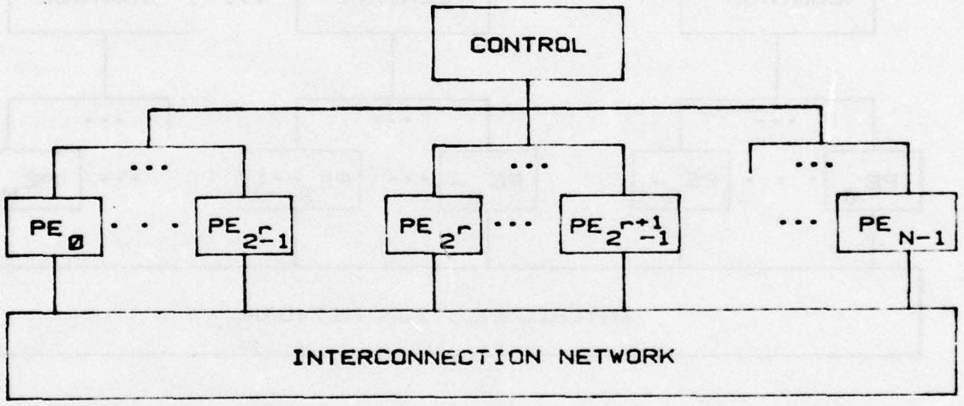


Figure V.1: For $N = 2^n$, single control partitioning supplies the same instruction stream to 2^{n-r} SIMD machines of size 2^r.

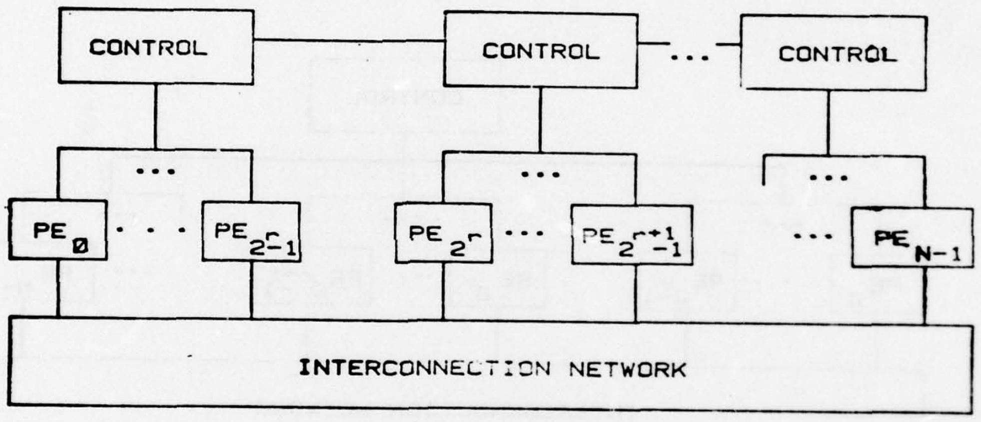


Figure V.2: For $N = 2^n$, multiple control partitioning implies multiple control units. Each of 2^{n-r} control units commands 2^r PEs. Two control units can be linked to form an SIMD machine of size 2^{r+1}.

section of the partitioned interconnection network must function as a complete network. That is, every section must have all of the capabilities of a network of the given type built for a size 2^r system.

This chapter considers partitioning interconnection networks using both single and multiple control partitioning. Section 2 considers multistage networks. Recirculating networks are analyzed in section 3. Section 4 analyzes pipelined multistage networks, and section 5 considers the use of variable size partitions, where all partitions are of size a power of two.

V.2. Partitioning Multistage Networks

This section shows how multistage Shuffle-Exchange, Cube, and PM2I networks can be partitioned.

Partitioning a multistage Shuffle-Exchange network (Figure III.3) is discussed in [LAS76]. That discussion is extended here and is related to the two types of partitioning defined above.

Theorem V.1: Let a system be partitioned into 2^{n-r} subsystems of 2^r PEs each. For single control partitioning, if only a shuffle and no exchange is allowed on each of the first n-r stages of the network, the network can serve as a complete r stage Shuffle-Exchange network for each of the 2^{n-r} groups of PEs of the partitioned machine.

Proof: Let the addresses of all PEs in each partition have the same n-r most significant bits. Let loc_i (P) refer to the location of the data originally in PE(P) after passing through i stages of the network [LAST76]. Let the destination address, the address of the PE which receives the data from P, be $D = d_{n-1}\ldots d_1 d_0$. After n-r shuffles with no exchanges or broadcasts (the first n-r stages of the network are in the straight state), the intermediate address of the data that was initially in P is

$$loc_{n-r}(P) = p_{n-(n-r)-1}\ldots p_1 p_0 p_{n-1}\ldots p_{n-(n-r)}$$
$$= p_{r-1}\ldots p_1 p_0 p_{n-1}\ldots p_r.$$

The next stage is the first that allows an exchange. So,

$$loc_{n-r+1}(P) = p_{r-2}\ldots p_1 p_0 p_{n-1}\ldots p_r d_{r-1},$$

where $d_{r-1} = \overline{p}_{r-1}$ or p_{r-1}. The final address of the data, after r Shuffle-Exchange steps, is

$$\mathrm{loc}_n\,(P) = p_{n-1}\cdots p_r d_{r-1}\cdots d_1 d_0,$$

where $d_i = \overline{p}_i$ or p_i. All PEs with addresses that have the same n-r most significant bits will be in the same partition of the machine. At stage i, $r < i \leq 0$, of the partitioned network, any interchange box compares data from two lines whose addresses differ only in the i^{th} bit position. If the n-r most significant address bits are ignored, these are the same two lines which are the inputs to the interchange box in a Shuffle-Exchange network for 2^r PEs. In this manner, a PE can communicate with any other PE in its partition, just as if it had a Shuffle-Exchange network of size 2^r. Also, no PE can disturb a PE not in its partition. Figure V.3 illustrates this partitioning for N = 16 PEs partitioned into four groups of four PEs each. []

An analogous argument may be made to show that the PEs may be partitioned based on any common set of n-r bits of the PE addresses.

Corollary V.1: Let a system be partitioned for single control partitioning into 2^{n-r} groups of 2^r PEs. Let all PEs in a group have some set of n-r address bits in common, and let these n-r bits be the same set of n-r bits for all groups. Then the network can serve as a complete r stage Shuffle-Exchange network for each of the 2^{n-r} groups of PEs of the partitioned machine.

Proof: After any stage of the Shuffle-Exchange network, exactly one bit of the intermediate address of the data can be changed. Stage n-1 affects only address bit n-1, so that after stage n-1,

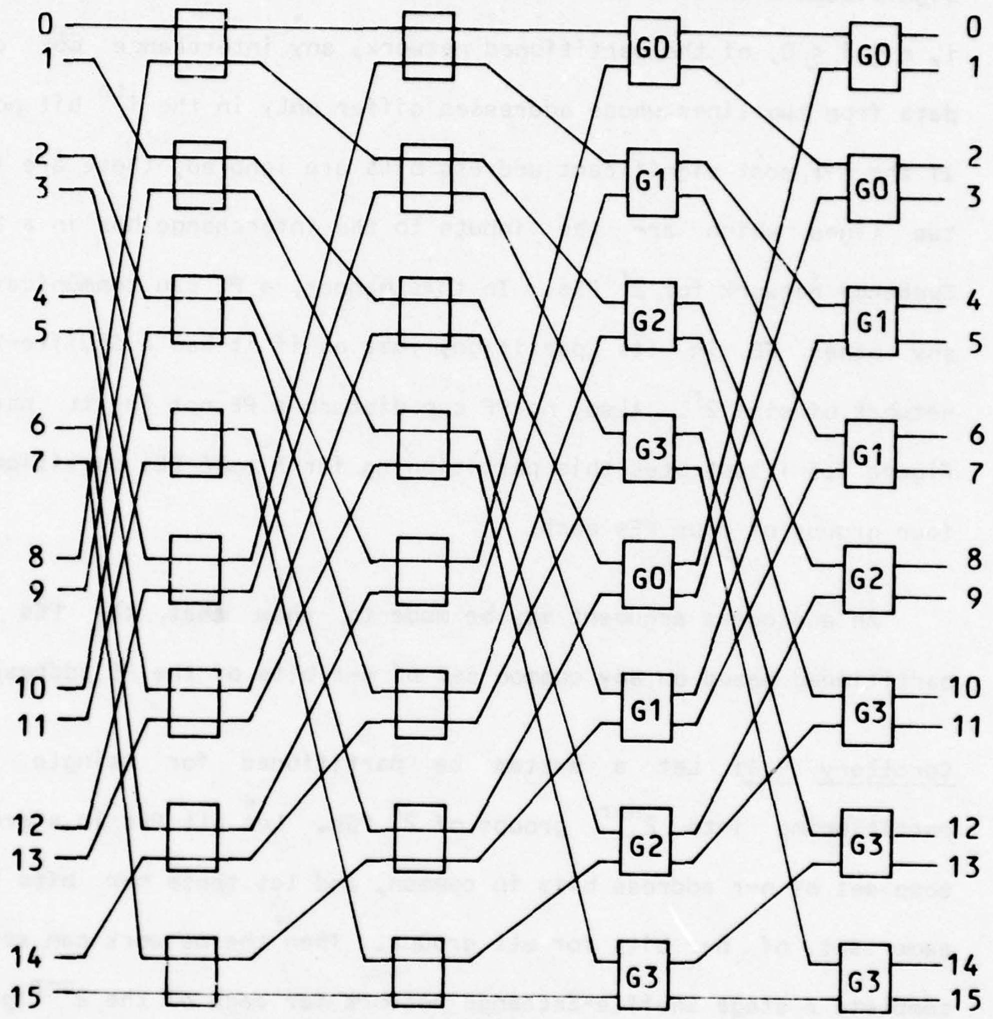


Figure V.3: For N = 16, a multistage Shuffle-Exchange network can be partitioned into four groups of four PEs such that for all PEs in a group, the two most significant PE address bits are the same.

$$loc_{n-1}(P) = p_{n-2} \cdots p_1 p_0 d_{n-1},$$

where $d_{n-1} = \overline{p}_{n-1}$ or p_{n-1}. Stages n-2 through 0 affect the intermediate address location of the data in analogous manners; stage i affects address bit p_i and changes it to d_i.

Let the n - r common PE address bits within all partitions be $p_{i_{n-r}}, \ldots, p_{i_1}$, where $i_j \in \{0, 1, \ldots, n-1\}$, $1 \le j \le n-r$. Since there is a one to one correspondence between the stage number and the address bits that stage changes, the network can be partitioned on any set of address bits. If p_{i_j} is one of the n-r common PE address bits, then at stage i_j, the exchange function is disallowed. []

If the Shuffle-Exchange network employs individual box control, both single control partitioning and multiple control partitioning can be supported by the network. For example, consider a three stage Shuffle-Exchange network with two function interchange boxes that is to be partioned into two groups of four PEs each. Let the two groups be G1 = (0, 1, 2, 3) and G2 = (4, 5, 6, 7). Consider the case where only G1 is using the network. Let c be a control matrix for the network of Figure V.4, which represents the control signals the interchange boxes receive. c(i,j) controls the i^{th} row interchange box at stage n-1-j. An exchange occurs at that module if and only if c(i,j) = 1. (This argument assumes two function boxes, but can be extended for four function boxes.) To control the network for G1, the matrix is

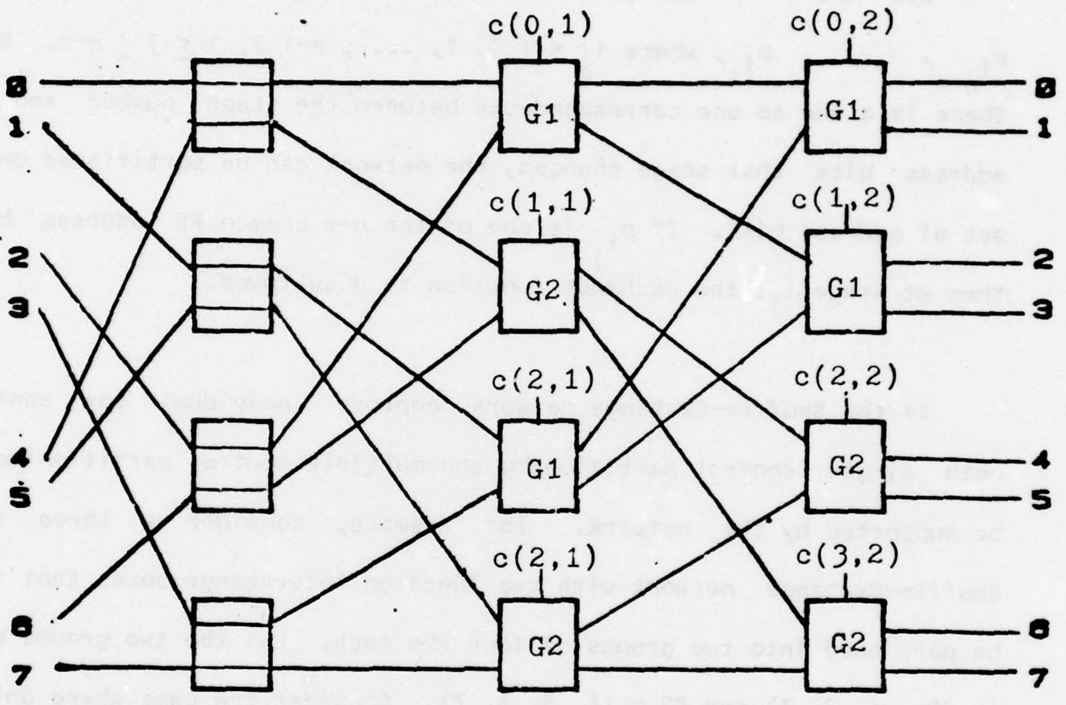


Figure V.4: A multistage Shuffle-Exchange network can be partitioned into two groups of four PEs each by setting one stage to no exchange. Group G1 is the four PEs 0, 1, 2, and 3. Group G2 is the four PEs 4, 5, 6, and 7.

$$c = \begin{bmatrix} 0 & c(0,1) & c(0,2) \\ 0 & - & c(1,2) \\ 0 & c(2,1) & - \\ 0 & - & - \end{bmatrix}$$

Only the control bits that relate to the submachine that is using the network need be specified. All others may be ignored. To control the network for G2 alone, the matrix is

$$c = \begin{bmatrix} 0 & - & - \\ 0 & c(1,1) & - \\ 0 & - & c(2,2) \\ 0 & c(3,1) & c(3,2) \end{bmatrix}$$

If G1 and G2 are to share the network under a single control partitioning scheme, set c(0,1) = c(1,1), c(2,1) = c(3,1), c(0,2) = c(2,2), and c(1,2) = c(3,2). For multiple control partitioning, G1 is controlled by c(0,1), c(2,1), c(0,2), and c(1,2), while G2 is controlled by c(1,1), c(3,1), c(2,2), and c(3,2). Thus, if c(i,0) = 0, $0 \leq i < 4$, the two partitions can operate independently and asynchonously.

For four function interchange boxes, the same c(i,j) control G1 and G2. For four function boxes, however, c(i,j) ∈ { 0, 1, 2, 3 }, where c(i,j) = 0 means straight, c(i,j) = 1 means exchange, c(i,j) = 2 means lower broadcast, and c(i,j) = 3 means upper broadcast.

<u>Corollary</u> <u>V.2</u>: An n stage Shuffle-Exchange (omega) network with individual box control c(i,j) can be partitioned in a multiple control environment, if all partitions are of size 2^r.

<u>Proof</u>: Divide the 2^n PEs into 2^{n-r} groups of 2^r, so that the n-r most significant address bits, $p_{n-1}\cdots p_r$, of all PEs in group G are the same. Since, the first n-r stages allow only a shuffle and no exchange, the entries of the first n-r columns of c are all 0, and

$$\mathrm{loc}_{n-r}(P) = p_{r-1}\cdots p_1 p_0 p_{n-1}\cdots p_r.$$

Stage r-1 is the first to allow an exchange to follow the shuffle. Let c(A, n-r) be a control bit for stage r-1 where $A = a_{n-2}a_{n-3}\cdots a_1 a_0$, since there are 2^{n-1} rows of c. Then c(A, n-r) controls PEs belonging to G if

$$A = a_{n-2}\cdots a_{n-r}p_{n-1}\cdots p_r.$$

That is, A is the address of an interchange box, $0 \leq A < n/2$. At stage r-1, interchange box A affects PEs of group G if and only if the last n-r address bits of A equal the first n-r address bits of any PE in G.

At stage r-(1+j), $1 \leq j < r$, c(A, n-r+j) controls PEs of G if

$$A = a_{n-2}\cdots a_{n-r+j}p_{n-1}\cdots p_r a_{j-1}\cdots a_0.$$

For single and multiple control partitioning, if the group of PEs is known, the control signals that refer to that group are given by the above expression. The expression can be extended for any set of common n-r address bits used for partitioning and for four function interchange boxes. A broadcast function sends one datum to both outputs of the interchange box. The effects are the same as sending data first to the output for the "straight across" setting, the address of which is in the partition, and then to the output for the "exchange" setting, which is also in the partition. That is, for a broadcast, both outputs are part of the same partition, and no data is transferred outside its partition.

$\square$

In [SIS78], it is shown that the indirect binary n-cube and the generalized cube are both topologically equivalent to the multistage Shuffle-Exchange network. Since these three networks have equivalent topologies, the partitioning proofs are similar.

Corollary V.3: The indirect binary n-cube network and the generalized cube network with individual box control can be partitioned in a multiple control environment if all partitions are of size 2^r.

Proof: Corollary V.3 follows from the topological equivalence of the indirect binary n-cube, the generalized cube, and the multistage Shuffle-Exchange network, and from Theorem V.1. The only difference between these two networks and the omega network discussed in Corollary V.2 is the position of the control bits in the control matrix. For example, for N=8, consider G1 as defined above. For the generalized cube, the control matrix would be

$$c = \begin{bmatrix} 0 & c(0,1) & c(0,2) \\ 0 & c(1,1) & c(1,2) \\ 0 & - & - \\ 0 & - & - \end{bmatrix}$$

as shown in Figure V.5. For the indirect binary n-cube, if $c'(i,j) = c(i,n-1-j)$, then:

$$c' = \begin{bmatrix} c'(0,0) & c'(0,1) & 0 \\ c'(1,0) & c'(1,1) & 0 \\ - & - & 0 \\ - & - & 0 \end{bmatrix}$$

as shown in Figure V.6. In both cases, the stage which affects address bit p_2 is set to no exchange. So, data cannot be exchanged between a PE

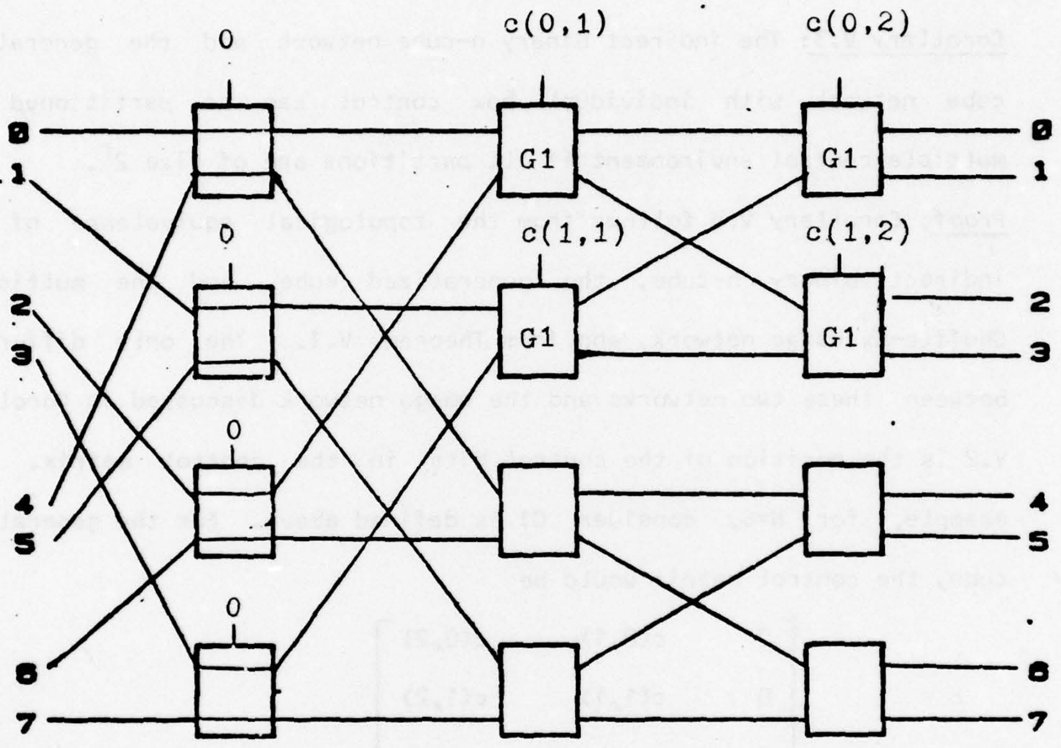


Figure V.5: If PEs 0, 1, 2, and 3 use the network, then a generalized cube network uses the controls above to affect data passage through the network.

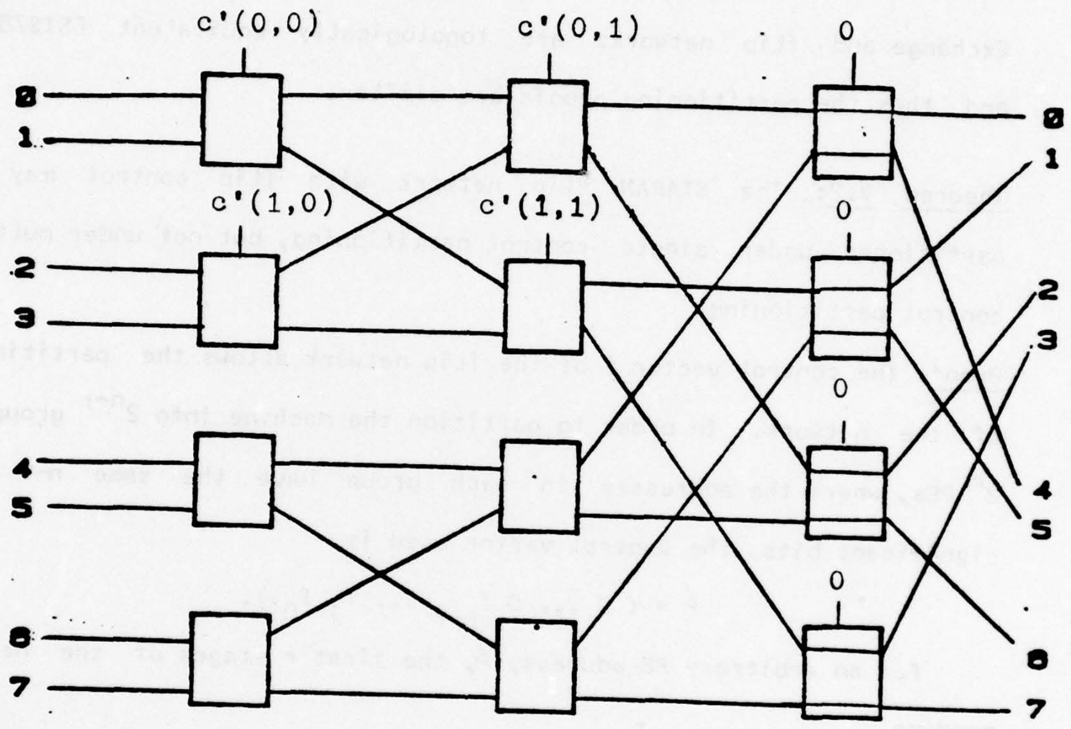


Figure V.6: An indirect binary n-cube network controls data from PEs 0, 1, 2, and 3 using the controls above.

in G1 with address 0XX and a PE in G2 with address 1XX, where, as in the PE address mask notation (Chapter II), X is "don't care," i.e., either 0 or 1. []

In the STARAN flip network [BA76] (Figure III.4), stage i can perform the Cube-type interconnection function Cube_i. The Shuffle-Exchange and flip networks are topologically equivalent [SIS78], and thus the partitioning proofs are similar.

Theorem V.2: The STARAN flip network with flip control may be partitioned under single control partitioning, but not under multiple control partitioning.

Proof: The control vector F of the flip network allows the partitioning of the network. In order to partition the machine into 2^{n-r} groups of 2^r PEs, where the addresses in each group have the same n-r most significant bits, the control vector used is

$$F = (0 \ldots 0 \, f_{n-r} \ldots f_1 \, f_0).$$

For an arbitrary PE address, P, the first r stages of the network produce

$$\text{loc}_r \, (P) = p_{n-1} \ldots p_r d_{r-1} \ldots d_0,$$

where $d_i = \overline{p}_i$ or p_i. If the last n-r stages allow no exchanges, the address of the final destination of the data is

$$\text{loc}_n \, (P) = p_{n-1} \ldots p_r d_{r-1} \ldots d_0.$$

Thus, all PEs with the same n-r most significant bits will be in the same partition of the reconfigurable machine. Again, this procedure can be generalized to group PEs that have any set of n-r bits in common. For example, if the n-r least significant addresses bits are the same for all PEs in a group, then

$$F = (f_{n-1} \dots f_{n-r}\ 0 \dots 0).$$

Since each stage of the flip network receives only one control bit, multiple control partitioning is not possible. []

Corollary V.4: The generalized cube network with individual stage control can be partitioned under single control partitioning, but not under multiple control partitioning.

Proof: Since the generalized cube network and the STARAN flip network are topologically equivalent [SIS78], Corollary V.4 follows from Theorem V.2. []

The STARAN also allows a set of "shift permutations." The shift controls for the STARAN provide i+1 control signals at stage i to move data at input x to output $x + 2^m$ modulo 2^p, $0 \leq m \leq p \leq n$. This set of controls is illustrated in Figure III.4b for N=8. In general, the controls at stage i, ci0, ci1, ..., and cii, affect groups of PEs as follows. Recall from Chapter II that a set of PE addresses can be expressed as a string of 0's, 1's, and X's, where X is a don't care, i.e., either 0 or 1. Also, X^i is a string of i X's in the specification of a PE address.

ci0: $X^{n-i}0^i$

ci1: $X^{n-i}0^{i-1}1$

ci2: $X^{n-i}0^{i-2}1X$

. . .

cii: $X^{n-i}1X^{i-1}$

Referring to Figure III.4b, c00 = 0A, c10 = 1A, c11 = 1B, c20 = 2A, c21 = 2B, and c22 = 2C. Under certain conditions, these controls can be effectively used for single control partitioning. However, they allow independent use for no groups of PEs if 2^{n-r} groups of 2^r PEs, $1 < r < n$, must use the network at one time under multiple control partitioning.

In general, the STARAN flip network with shift control cannot always be partitioned into 2^{n-r} groups of 2^r PEs using single control partitioning. Consider the case for N = 8 and r=2, where the two groups of PEs are G0 = { X0X } and G1 = { X1X }. Physical stage 0 is logical stage 0. Since the middle bit position, p_1, determines the partitioning, physical stage 1 must be set to straight in order to isolate the partitions. Physical stage 2 is logical stage 1. In a STARAN network for four PEs, logical c10 affects all PEs in a group whose addresses end in 0, i.e., PEs {XX0}, and logical c11 affects all PEs in a group whose addresses end in 1, i.e., PEs {XX1}. But, physical c22 affects all PEs X1X. It affects some that should respond to logical c10, i.e., PEs {X10}, and some that should respond to logical c11, i.e., PEs {X11}. So, single control partitioning is not possible for this case.

<u>Theorem V.3</u>: Let a system be partitioned into 2^{n-r} groups of 2^r PEs. If the n-r PE address bits which define a partition are the n-r most significant address bits or the n-r least significant address bits, then the STARAN flip network with shift controls can be partitioned under single control partitioning.

Proof: If the n-r most significant address bits determine the partitioning, then physical stage i is logical stage i, $0 \le i < r$. Stages r through n-1 are set to no exchange (straight through the stages). All control signals affect the same set of PEs of a partition. For example, at stage i, $0 \le i < r$, ci0 affects PEs with addresses $X^{n-i}0^i$. Since $n-i > n-r$, all PEs with addresses whose least significant bits are 0^i in all partitions are affected by ci0. ci1 affects PEs $X^{n-i}0^{i-1}1$. Since $n-i > n-r$, all PEs with addresses whose least significant bits are $0^{i-1}1$ in all partitions are affected by ci1. So, single control partitioning is possible.

If the partitioning is based on the n-r least significant PE address bits, then all PEs in a partition have the same n-r least significant bits. These n-r bits remain the same at each intermediate address location of the data as it passes through the network. So, stages 0 through n-r-1 are set to no exchange for all PEs. Physical stage n-r is logical stage 0 for the partitions. One logical control, c00, is needed, so all n-r+1 controls of the stage are set to c00.

Stage $k = n-r+1$ is logical stage 1. Note that $n - (n - r + 1) = r-1$. Logical control c10 affects PEs $X^{r-1}0$ of all partitions, that is, PEs $X^{r-1}0X^{n-r}$. The physical controls that become c10 and the PEs they affect are

ck0: $X^{r-1}00^{n-r}$

ck1: $X^{r-1}00^{n-r+1}1$

ck2: $X^{r-1}00^{n-r+2}1X$

. . .

ck(k-1): $X^{r-1}01X^{n-r-1}$.

Logical c11 affects PEs $X^{r-1}1X^{n-r}$. So, logical c11 is physical ckk.

For $i > 1$, logical stage i is physical stage r-r+i. Let k = n-r+i. Then n-k = n - (n-r+i) = r-i. Logical ci0 affects PEs $X^{r-i}0^i$ of all partitions, i. e., PEs $X^{r-i}0^iX^{n-r}$. The physical controls that become ci0 and the PEs they affect are

$$ck0\text{: } X^{r-i}0^i0^{n-r}$$

$$ck1\text{: } X^{r-i}0^i0^{n-r-1}1$$

$$\cdots$$

$$ck(k-i)\text{: } X^{r-i}0^i1X^{n-r-1}.$$

Logical controls cij, $0 < j < i$, are similar. Logical cii is physical control signal ckk.

For example, let N = 8 (as in Figure III.4b) and r = 2. The system is divided into $2^{3-2} = 2$ groups of $2^2 = 4$ PEs each. Physical stage 0 is set to straight. Logical stage 0 is physical stage 1, so logical control c00 is set to physical controls c10 and c11. Logical stage 1 is physical stage 2. Logical c10 affects the four PEs {X0X}. So, physical controls c20, (which affects PEs {X00}), and c21, (which affects PEs {X01}), become logical control c10. c11 affects PEs {X1X}, and so physical c22 becomes logical c11. []

<u>Theorem V.4</u>: The STARAN flip network with shift controls can not be partitioned under multiple control partitioning, if 2^{n-r} groups of 2^r, $1 < r < n$, use the network at one time.

<u>Proof</u>: For multiple control partitioning, if all groups are of size 2 and so require only one stage of the network, then 2^{n-1} groups of two PEs can independently use the network. This is because if two PEs whose

addresses differ in the i th bit position exchange data, then Cube_i is executed. If they do not exchange data, then they need not use the network.

Consider multiple control partitioning for 2^{n-r} groups of 2^r PEs, $1 < r < n$.

Case 1: 2 groups of 2^{n-1} PEs ($r = n-1$).

Here, n-1 stages of the STARAN network are needed so that each group may have a complete network to use. Since stage 0 has only one control, the n-1 stages must be stages 1 through n-1. Stage 1 supplies one control signal for each group, and so is logical stage 0 for each partition. Stage 2, however, has only three controls, not the necessary four (two for each partition). So, 2 groups of 2^{n-1} cannot use the network independently.

Case 2: 2^{n-r} groups of 2^r, $1 < r < n-1$.

Logical stage 0 must have 2^{n-r} independent controls, one for each group of 2^r PEs. Suppose physical stage i is logical stage 0 for each group of the partition. Stage i must have $i+1 \geq 2^{n-r}$ independent controls, each control affecting 2^r PEs. But, cii affects PEs $X^{n-i}1X^{i-1}$, i.e., 2^{n-1} PEs. Since $2^{n-1} > 2^r$, $1 < r < n-1$, stage i does not provide enough controls so that each group of PEs may use the network independent of any other group. []

Corollary V.5: The generalized cube with i+1 control signals at stage i can be partitioned under single control partitioning, if the PE address bits which define the partition are the n-r most significant bits or the n-r least significant bits. Also, the network cannot be partitioned under multiple control partitioning.

Proof: Since the generalized cube is topologically equivalent to the STARAN flip network, this corollary follows from Theorem V.3 and Theorem V.4. []

The data manipulator [FE74] is an n stage network where stage i can perform $PM2_{+i}$ (D), $PM2_{-i}$ (U), and no change (H), as shown in Figure III.7. Each control is further divided into 2 sets as shown. For stage i, U_1, D_1, and H_1 control those PEs whose i^{th} bit is 0, and U_2, D_2, and H_2 control those PEs whose i^{th} bit is 1.

This multistage network may easily be divided into 2^{n-1} groups of 2 adjacent PEs (N' = 2). Let the two PEs be PE(i) and PE(i+1 modulo N'), for i even. Since N' = 2, the network need consist only of $PM2_{\underline{+}0}$ (note that for N' = 2, $PM2_{+0}$ is the same as $PM2_{-0}$). This can be obtained using D_1U_2 on stage 0 and H_1H_2 on all other stages. Similarly, PE(i) and PE(i+1 modulo N'), for i odd, can exchange data if U_1D_2 is used on stage 0 and H_1H_2 is used on all other stages. This is true for multiple control partitioning since PEs not exchanging data can ignore the network.

As an additional point, consider pairing two arbitrary PEs.

<u>Theorem V.5</u>: For the data manipulator network, for the control functions given as in Figure III.7, if only two PEs use the network at one time (single control partitioning), then any two PEs can exchange data in one pass through the network.

<u>Proof</u>: For N PEs, let PE A and PE B exchange data, where $A = a_{n-1} \cdots a_0$, $B = b_{n-1} \cdots b_0$, and $A \neq B$. In order for PE A and PE B to exchange data, the destination address of the data from A is $a'_{n-1} \cdots a'_0 = b_{n-1} \cdots b_0$, and the destination address of the data from B is $b'_{n-1} \cdots b'_0 = a_{n-1} \cdots a_0$. An algorithm to perform such an input address to output address translation is as follows.

```
For stage i, $0 \leq i < n$ :
    If $a_i = b_i$ , then no change ($H_1 H_2$).
    If $a_i = 0$ and $b_i = 1$, then apply $PM2_{+i}$ to the
        data from A, and apply $PM2_{-i}$ to the data from
        B ($D_1$ $U_2$). Thus,
            $a'_i = a_i + 1 = 1 = b_i$, and
            $b'_i = b_i - 1 = 0 = a_i$.
    If $a_i = 1$ and $b_i = 0$, then apply $PM2_{-i}$ to the
        data from A, and apply $PM2_{+i}$ to the data from
        B ($D_1$ $U_2$). Thus,
            $a'_i = a_i - 1 = 0 = b_i$, and
            $b_i' = b_i + 1 = 1 = a_i$.
```

For this algorithm, after the i th stage,

$$\mathrm{loc}_i\,(A) = b_{n-1}\ldots b_{n-i}a_{n-i-1}\ldots a_0, \text{ and}$$

$$\mathrm{loc}_i\,(B) = a_{n-1}\ldots a_{n-i}b_{n-i-1}\ldots b_0.$$

Finally, after n stages,

$$\mathrm{loc}_n\,(A) = b_{n-1}\ldots b_1b_0, \text{ and}$$

$$\mathrm{loc}_n\,(B) = a_{n-1}\ldots a_1a_0.$$

Table V.1 shows the control settings for stage i that result from the execution of the algorithm above. []

This algorithm may be adapted for both the n stage Shuffle-Exchange and the flip networks. Let PEs P and Q exchange data. For the Shuffle-Exchange, if bit i of the two PE addresses are the same, no Exchange occurs at stage i, and $c(A, n-1-i) = 0$, $0 \leq A < 2^{n-1}$. If the two bits differ, then an exchange occurs, and $c(A, n-1-i) = 1$, $0 \leq A < 2^{n-1}$. For the flip network, at stage i, if the i th bits of P and Q are the same, then $f(i) = 0$, and no exchange occurs. If the two bits differ, then $f(i) = 1$, and an exchange occurs.

If a restriction is made, the data manipulator network can be partitioned such that each group has a complete data manipulator network at its disposal. For example, let N = 8, and let the machine be partitioned into 2^1 groups of 2^2 PEs. Let one group contain PEs 0, 2, 4, and 6, while the other group contains PEs 1, 3, 5, and 7. The control structure of this network is not general enough to allow multiple control paritioning, and so only single control partitioning will be considered. If only the straight through path (H_1 H_2) is allowed at stage 0, then each group of 4 PEs has a $\log_2$ 4 stage data

Table V.1: The data manipulator network can exchange two data items if the following controls are applied at stage i, $0 \leq i < n$.

i^{th} bit of A	i^{th} bit of B	Control setting for stage i of the data manipulator network.
0	0	H_1 H_2
0	1	D_1 U_2
1	0	D_1 U_2
1	1	H_1 H_2

manipulator network available to it. The physical stage 1 now corresponds to a logical stage 0 for each partitioned group, since each PE address differs from the addresses of its neighbors by two modulo 8. $PM2_{+1}$ moves data from position 0 to position 2, from 2 to 4, from 4 to 6, and from 6 to 0. Similarly, the physical stage 2 corresponds to a logical stage 1 for the group. Figure V.7 illustrates this process.

Theorem V.6: Let the data manipulator be partitioned into 2^{n-r} groups of 2^r PEs so that all PEs in a partition have the same n-r low order bits. Under this restriction, the data manipulator can be partitioned under single control partitioning.

Proof: Each group is composed such that consecutive PEs within a group have PE addresses that differ by 2^{n-r}. Then, stage n-r behaves as a stage 0 for the group, stage (n-r+1) behaves as a stage 1, and so on, and lastly, stage n-1 behaves as a stage (r-1) for the group of 2^r PEs. Stage i, $0 \leq i < n-r$, must be set in the "H" state so that address bits p_{n-r-1} through p_0 are the same for both the input address and the output address of the data passing through the network. For all j, $PM2_{\pm j}(p_{n-1}\cdots p_0) = p'_{n-1}\cdots p'_j p_{j-1}\cdots p_1 p_0$. That is, adding or subtracting one in bit position j cannot affect bit position i, $0 \leq i < j$, of a PE address. So stage j, $n-r \leq j < n$, cannot affect address bit i, $0 \leq i < n-r$, and the partitions are independent. □

Corollary V.6: Let the N PEs be partitioned as in Theorem V.6. Then the augmented data manipulator (ADM) can be used under single or multiple control partitioning, if all partitions are the same size.

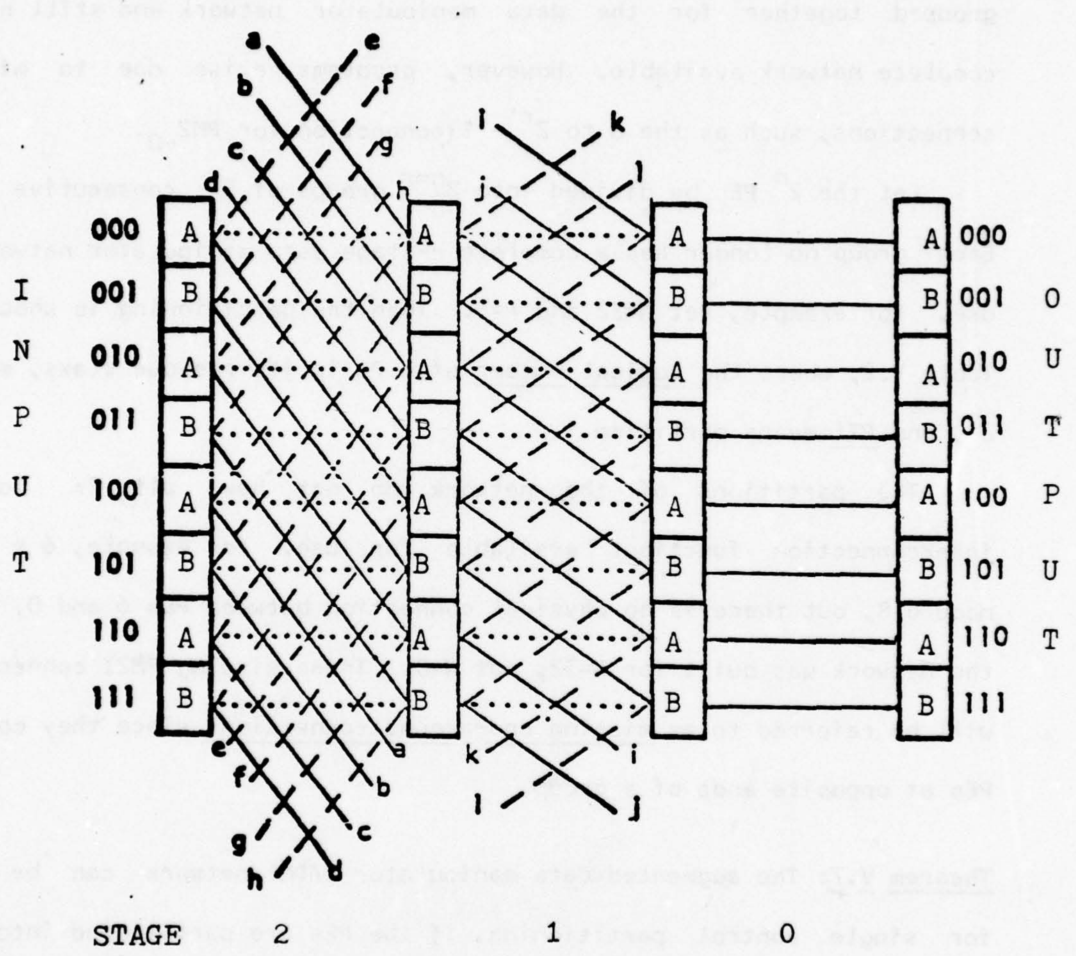


Figure V.7: For N = 8, a multistage PM2I network can be partitioned into two groups of four PEs. Group A is PEs 0, 2, 4, and 6. Group B is PEs 1, 3, 5, and 7.

Proof: Follows from Theorem V.6 and the definition of the ADM network.[]

There are means of allowing 2^r PEs with consecutive addresses to be grouped together for the data manipulator network and still have a complete network available. However, problems arise due to missing connections, such as the 0 to $2^r - 1$ connection for $PM2_{-0}$.

Let the 2^n PEs be divided into 2^{n-r} groups of 2^r consecutive PEs. Each group no longer has a complete r-stage data manipulator network to use. For example, let N=32 and r=3. Then the partitioning is shown in Table V.2, where the logical number of a PE is its residue class, modulo 2^r, and PTi means partition i.

The partitions of the network do not have all 2r logical interconnection functions available for use. For example, 6 + 2 = 0 modulo 8, but there is no physical connection between PEs 6 and 0, since the network was built for N=32, not N=8. These missing PM2I connections will be referred to as missing end-around connections since they connect PEs at opposite ends of a group.

Theorem V.7: The augmented data manipulator (ADM) network can be used for single control partitioning, if the PEs are partitioned into 2^{n-r} groups of 2^r consecutive PEs.

Proof: Consider the case where the virtual $PM2_{+i}$ interconnection function, for some $0 \leq i < r$, is a missing end-around connection for the data item I that is in cell y just prior to stage i (the case for $PM2_{-i}$ is similar). Let cell y be the b^th cell in the p^th partition of size 2^r, that is:

$$y = p2^r + b, \quad 2^r - 2^i \leq b < 2^r, \quad 0 \leq p < 2^{n-r}.$$

Table V.2: Partitioning based on consecutive PEs using the augmented data manipulator network.

Logical	Physical number				Notes
number	PT0	PT1	PT2	PT3	
0	0	8	16	24	need $PM2_{-j}$, j = 0,1,2.
1	1	9	17	25	need $PM2_{-j}$, j = 1,2.
2	2	10	18	26	need $PM2_{-j}$, j = 2.
3	3	11	19	27	
4	4	12	20	28	
5	5	13	21	29	need $PM2_{+j}$, j = 2.
6	6	14	22	30	need $PM2_{+j}$, j = 1,2.
7	7	15	23	31	need $PM2_{+j}$, j = 0,1,2.

If $PM2_{+i}$ is a missing end-around connection, then $b \geq 2^r - 2^i$. $PM2_{+i}$ sends this data to cell

$$y1 = y + 2^i = (p2^r + b + 2^i) \text{ modulo } N$$
$$= (((p + 1) \bmod 2^{n-r})2^r + b') \text{ modulo } N,$$
$$\text{where } b' = b + 2^i - 2^r, \quad 0 \leq b' < 2^i.$$

Thus, the data is in cell b' of partition p+1 modulo 2^{n-r} instead of cell $b+2^i$ modulo 2^r of partition p. Figure V.8 illustrates the errant data movement for $N = 8$ and $2^r = 4$. When the $PM2_{+1}$ function is applied to PE(2) of partition 0, data moves to PE(4), which is logical PE(0) of partition 1. The correct data movement would be to PE(0) of partition 0.

Since $2^r - 2^i \leq b < 2^r$ and $i < r$,

$$(b + 2^i) \text{ modulo } 2^r = (b + 2^i - 2^r) \text{ modulo } N.$$

Thus,

$$y1 - 2^r \text{ modulo } N = p2^r + (b + 2^i \text{ modulo } 2^r) \text{ modulo } N.$$

If stages i-1 through 0 are "H" for data item I, then if the network is set so that stage r is "U" for item I, the missing end-around connection will be compensated for before it occurs. Figure V.9 shows the correction for the example of Figure V.8. The flow of data is from physical PE(2) (logical PE(2) of partition 0) to PE(6) = PE($PM2_{-2}$ (2)) to PE(0) = PE($PM2_{+1}$ (6)).

If $PM2_{-i}$ is the missing end around connection, then an argument similar to the one above shows that setting stage r to $PM2_{+r}$ for the appropriate cells corrects any false data movement.

Consider the case where after passing through the rest of the network, data item I is moved, after stage i, from cell y1 to y1 + d, $-2^i < d < 2^i$, i.e., stages i-1 through 0 are not all "H" for item I. As

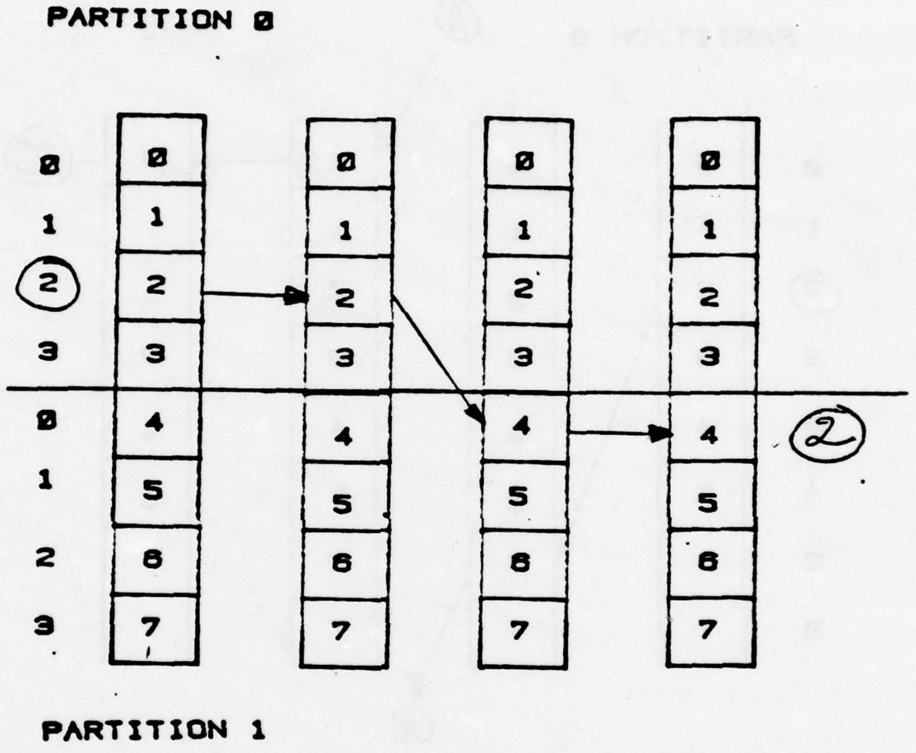


Figure V.8: $PM2_{+1}$ is a missing end-around connection for PE(2) of partition 0. It moves data from PE(2) to PE(4) in partition 1 instead of moving the data to PE(0) of partition 0.

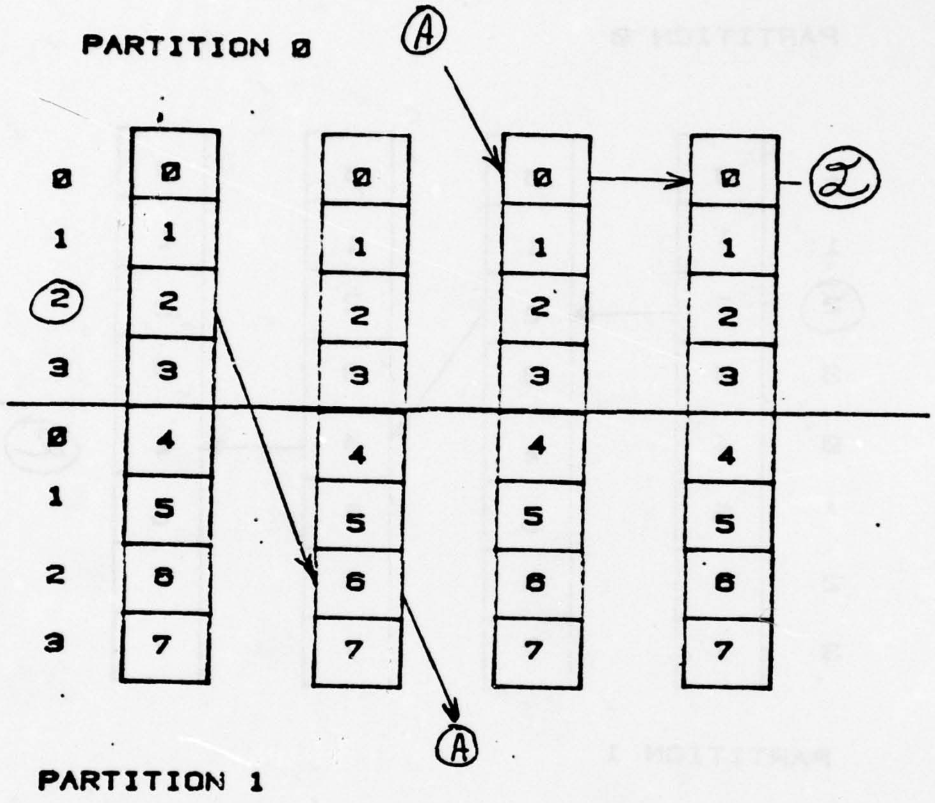


Figure V.9: Applying $PM2_{-2}$ to PE(2) before applying $PM2_{+1}$ corrects for the missing end-around $PM2_{+1}$ connection for PE(2).

opposed to the d=0 case above, when $-2^i < d < 2^r - 2^i - b$ no compensation is needed, i.e.,

$$y1 + d \text{ modulo } N = (p2^r + b + 2^i + d) \text{ modulo } N$$
$$= (p2^r + (b + 2^i + d) \text{ modulo } 2^r) \text{ modulo } N,$$
$$2^r - 2^i \leq b < 2^r,\ i < r.$$

Figure V.10 illustrates the above case. In Figure V.8, the $PM2_{+1}$ interconnection function moved data from PE(2) to PE(4). If interconnection function $PM2_{-0}$ is applied to PE(4), then the data moves to PE(3) of partition 0. That is, the net distance the data from PE(2) moved was (+ 2 - 1) = +1, i.e., from PE(2) to PE(3) of partition 0.

When $2^r - 2^i - b \leq d < 2^i$, the only compensation that is needed is a single -2^r, as in the d=0 case above. Since

$$(y1 + d - 2^r) \text{ modulo } N = (p2^r + b + 2^i + d - 2^r) \text{ modulo } N.$$

Then, $2^r - 2^i - b \leq d < 2^i$, so $0 \leq b+2^i+d-2^r < 2^{i+1} - 2^r + b$. Furthermore, since $2^r - 2^i \leq b < 2^r$, then $0 \leq b + 2^i + d - 2^r < 2^{i+1}$, where $i < r$. Thus, $b + 2^i + d - 2^r$ modulo $N = b + 2^i + d$ modulo 2^r. As an example, suppose that it is desired to move data from PE(2) to PE((2 + 3) modulo 4 = 1) of partition 0 by applying the virtual interconnection functions $PM2_{+1}$ and $PM2_{+0}$. Setting stage r to $PM2_{-2}$ for PE(2) causes the desired movement of data. Figure V.11 illustrates the correction.

The above techniques of grouping consecutive PEs are limited to single control partitioning, since data will cross partition boundaries.

In summary, for multiple control partitioning, the data manipulator network is adequate to divide the PEs into 2^{n-1} groups of two PEs where

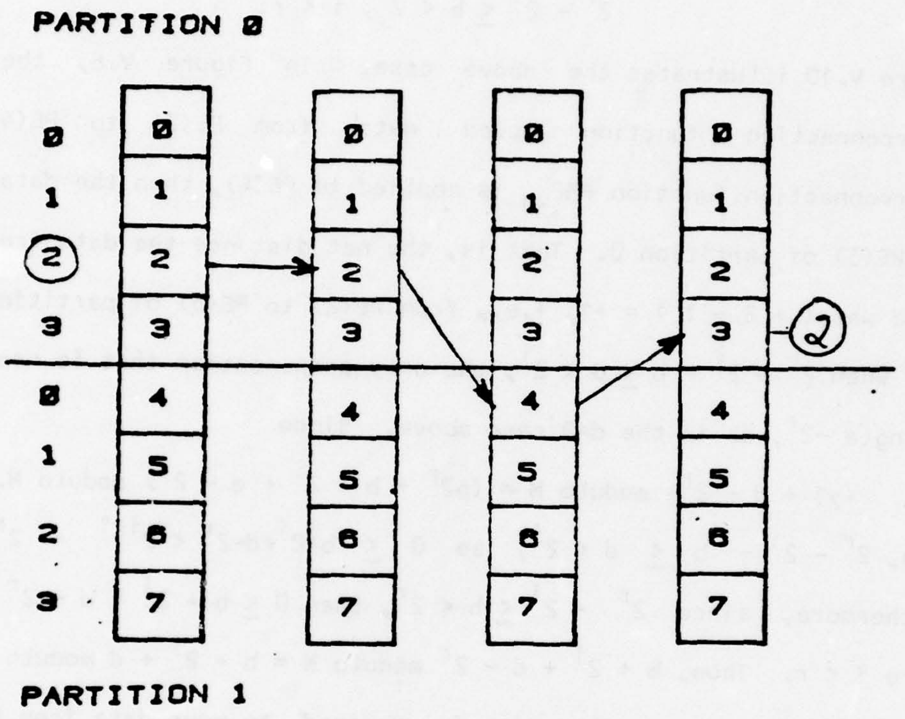


Figure V.10: A missing end-around $PM2_{+1}$ connection for PE(2) is corrected by a missing end-around $PM2_{-0}$ connection applied to the data from PE(2).

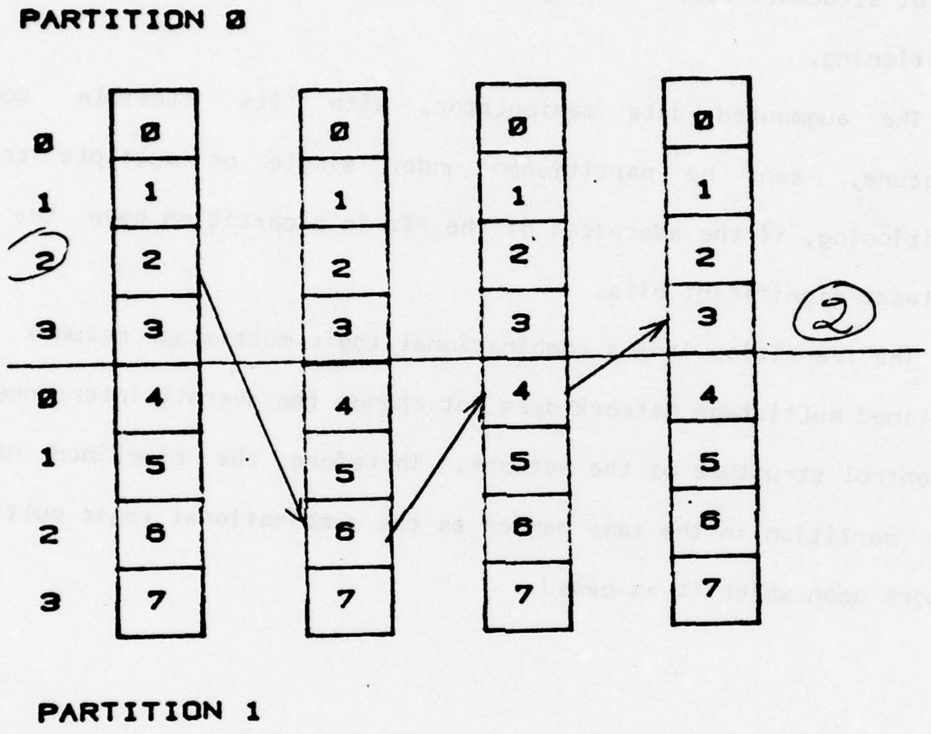


Figure V.11: Applying $PM2_{-2}$ to PE(2) before applying $PM2_{+1}$ and $PM2_{+0}$ causes data from PE(2) to be sent to PE(2 + 2 + 1 modulo 4) = PE(1) of partition 0.

a group consists of PE(i) and PE(i+1 modulo N), $0 \leq i < N$. With single control partitioning, certain sets of 2^r PEs may have an r stage network available, but the grouping of 2^r PEs is more restricted than with the STARAN flip and Shuffle-Exchange networks. Like the flip network, the control structure here is not general enough to allow multiple control partitioning.

The augmented data manipulator, with its flexible control structure, can be partitioned under single or multiple control partitioning, if the addresses of the PEs in a partition have the same n-r least significant bits.

The transition from a combinational logic multistage network to a pipelined multistage network does not change the overall interconnection or control structure of the network. Therefore, the pipelined network must partition in the same manner as the combinational logic multistage network upon which it is based.

V.3. Partitioning Recirculating Networks

Section V.3 considers partitioning the recirculating Illiac, Cube, and PM2I networks. Both single and multiple control partitioning are considered.

To control the Illiac network, let us assume that the control unit broadcasts to all active PEs one of the four functions to execute as they would any other instruction. Then, some partitions are easily implemented with this network.

Theorem V.8: An Illiac network may be partitioned into 2^{n-1} groups of 2 PEs, where the two are PE(i) and PE(i+1) or PE(i) and PE(i+m).

Proof: Let the group be PE(i) and PE(i+1), for either i even or i odd. PE(i) needs only Illiac_{+1} and PE(i+1) needs only Illiac_{-1} in order to move data about the submachine. Let the group be PE(i) and PE(i+m). Then, PE(i) uses Illiac_{+m} and PE(i+m) uses Illiac_{-m} to move data about the network. []

The Illiac network cannot be partitioned into groups of 4 without some restrictions. If PE(i), PE(i+1), PE(i+m), and PE(i+1+m) are grouped together, Figure V.12 shows their configuration for N = 16. Note that if a complete Illiac network was available for a system of four PEs, as in Figure V.13, then PE(i) and PE(i+1+m) would be connected, and PE(i+1) and PE(i+m) would be connected. In Figure V.12, they are not. The partitioned network cannot pass data in the same way as can a complete Illiac network for N = 4. If, however, by appropriately controlling the active status of the PEs, we allow PE(i)

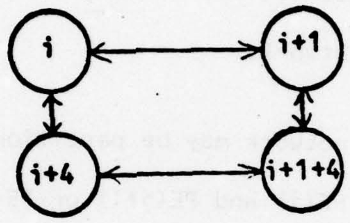


Figure V.12: Four PEs out of 16 grouped by an Illiac connection.

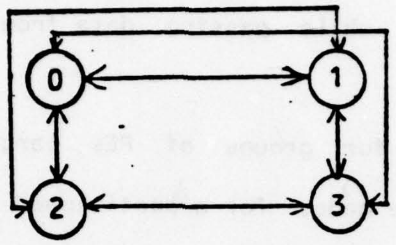


Figure V.13: An Illiac network for N = 4, m = 2.

to first pass data to PE(i+1), and then let PE(i+1) pass this data to PE(i+1+m), the appropriate connection between PE(i) and PE(i+1+m) can be realized. The transfer of data now requires that the interconnection network be used twice, and thus the time to pass the data is doubled. Also, steps must be taken to insure that the data presented to PE(i+1) is not destroyed while passing data from PE(i) to PE(i+1+m) through PE(i+1).

In general, for groups of PEs larger than two, this same observation may be made. For a partitioned machine of size 2^r, the PEs at the side edges of the array (i.e., PE(0), PE($2^{r/2}$ -1), PE($2^{r/2}$), PE(2($2^{r/2}$) - 1), . . . , and PE(2^r - 1)), and the top and bottom edges will not be connected as they would with a complete Illiac network for N = 2^r.

The Illiac network may easily exchange data between two adjacent PEs. For larger groups of PEs, the data may need to be passed among PEs to simulate the edge connections of the Illiac. This results in additional considerations for the control of the network, additional time for the data to reach their final destinations, and additional pitfalls for the programmer. The problem occurs in both multiple and single control partitioning, since the edge connections for a group can be simulated using only the PEs within that group.

In chapter IV, a circuit was presented for a recirculating Cube network. At each pass through the network, any of the n Cube interconnection functions may be selected to route data.

<u>Theorem V.9</u>: A recirculating Cube network can be partitioned into 2^{n-r} groups of 2^r PEs under single or multiple control partitioning, if all partitions are of size 2^r, $0 \leq r \leq n$.

<u>Proof</u>: A recirculating Cube may be partitioned in a manner analogous to that of a multistage Cube. Let the addresses of all PEs in each partition have the same n-r most significant bits. In order to partition the recirculating Cube network, the n-r interconnection functions Cube_r, Cube_{r+1}, ..., and Cube_{n-1} are not utilized for any data transfer. So, each group of 2^r PEs has available to it Cube_{r-1}, ..., Cube_1, and Cube_0. Thus, single and multiple control partitioning are possible for groups of size 2^r.

The above argument can be extended to group PEs that have any set of n-r bits in common. []

Chapter IV introduced a circuit for a recirculating PM2I network. Any of the 2n - 1 distinct PM2I interconnection functions may be chosen at each use of the network.

<u>Theorem V.10</u>: A recirculating PM2I network may be partitioned, under single or multiple control partitioning, into 2^{n-r} groups of 2^r PEs so that all PEs in a partition have the same n-r least significant bits.

<u>Proof</u>: Theorem V.6 showed how a multistage PM2I network can be partitioned into groups that have the n-r least significant bits in common by not utilizing the last n-r stages of the network, that is, the stages for $\text{PM2}_{\pm i}$, $0 \leq i < n-r$. In this manner, each group has available to it 2*r interconnection functions, where $\text{PM2}_{\pm(n-r+i)}$ now behaves as $\text{PM2}_{\pm i}$ for a group, $0 \leq i < r$. No group can send data outside of its

group, since using these 2*r functions, the n-r least significant address bits never change. That is, for all PE addresses P = $p_{n-1}...p_1p_0$,

$$PM2_{+j}\ (P) = p'_{n-1}...p'_j p_{j-1}...p_1p_0,$$

$n-1 \leq j \leq n-r$. So, each group may use the network independently of any other group. Thus, single and multiple control partitioning are possible for partitions of size 2^r. []

V.4. Multiple Control Partitioning and Variable Size Partitions.

For multiple control partitioning, partitioning the machine into groups of PEs of equal size has been considered. Some networks can be partitioned using a less restrictive multiple control partitioning where the only constraint is that all partitions be of size a power of 2.

Corollary V.7: If a network can be partitioned using restricted multiple control partitioning, where all partitions are the same size, then it can also be partitioned such that groups are of different sizes, where all sizes are powers of two.

Proof: Corollaries V.2, V.3, and V.6 can be applied recursively to break large partitions into smaller ones. Let the network be partitioned into 2^{n-r} groups of 2^r PEs. For any one of these 2^r groups, the applicable corollary may be applied again to break the group into 2^{r-k} groups of 2^k PEs, $0 \leq k < r$. Figure V.14 shows a generalized cube for N = 8 which is partitioned into three groups, G0, G1, and G2. G0 and G1 are of size 2. G2 is of size 4. □

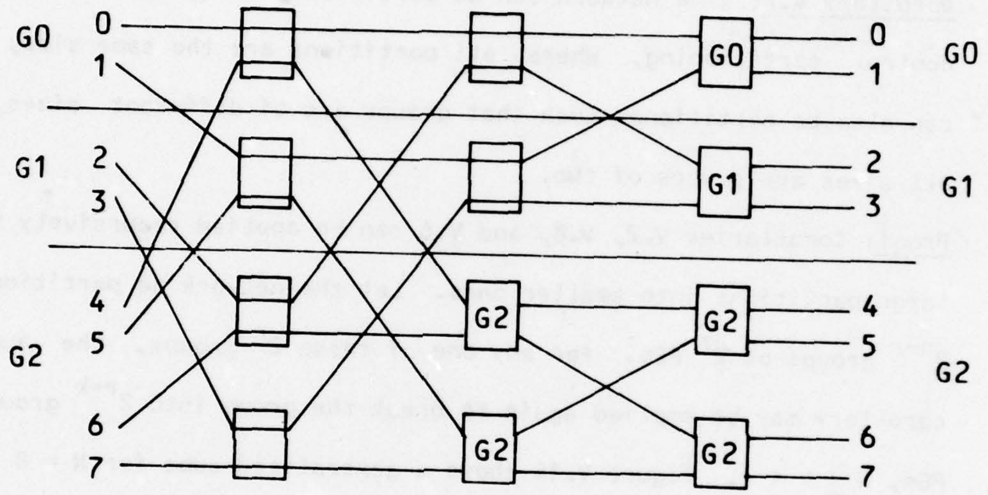


Figure V.14: A generalized cube can be partitioned into groups of different sizes when all sizes are powers of two. Here, G0 and G1 are of size 2, and G2 is of size 4.

V.5. Conclusions

Two ways have been defined to partition an SIMD machine consisting of 2^n processing elements into two to 2^{n-r} groups of 2^r PEs, each group acting as an SIMD machine of size 2^r. One way, single control partitioning, used a single control to broadcast instructions to all groups. Thus, the same SIMD machine program may be executed on several different data sets simultaneously (e.g., multiplying two pairs of matrices). The other way, multiple control partitioning, assumes the availability of additional control units so that each 2^r PE submachine may have its own instruction stream. The data being acted on by different submachines may be copies of the same data file (e.g., processing two copies of the same image to find different features).

Under single control partitioning, the Shuffle-Exchange network, the STARAN flip network, the generalized cube network, and the indirect binary n-cube network are readily partitioned into 2^{n-r} groups of 2^r PEs. The data manipulator network and the augmented data manipulator can group certain sets of 2^r PEs, but the ability to connect sets of 2^r PEs with consecutive addresses is restricted.

The n stage Shuffle-Exchange network, the generalized cube network, the indirect binary n-cube network, and the ADM network have control structures that are general enough to accommodate multiple control partitioning. The data manipulator, the flip network, the indirect binary n-cube, and the Shuffle-Exchange network can connect any two arbitrary PEs in one pass through the network.

A pipelined multistage network can be partitioned in the same manner as the combinational logic multistage network on which it is based. Recirculating Cube and PM2I networks can be partitioned in the same fashion as the multistage Cube and PM2I networks. Table V.3 summarizes the results of Chapter V.

Chapter V has defined the ways in which certain interconnection networks can be partitioned. This information is essential for selecting an interconnection network for a partitionable parallel processing system. In addition, the results of chapter V are relevant to the theory of the types of permutations from input addresses to output addresses that a given type of interconnection network with a given control can and cannot accomplish.

Table V.3: Summary of the partitioning properties of the networks discussed in Chapter V. $N = 2^n$ PEs are partitioned into 2^{n-r} groups of 2^r PEs, $0 < r < n$.

NETWORK	PARTITIONING
Multistage Shuffle-Exchange, omega, generalized cube, indirect binary n-cube	Any set of n-r common PE address bits defines a partition for either single or multiple control partitioning.
Flip network, flip control	For single control partitioning, any set of n-r PE address bits defines a partition. The flip network with flip control cannot be partitioned under multiple control partitioning.
Flip network, shift control	For single control partitioning, the n-r least significant PE address bits or the n-r most significant address bits define a partition. Multiple control partitioning is not possible.
data manipulator	The n-r least significant PE address bits define a partition.
Augmented data manipulator	The n-r least significant PE address bits define a partition. Also, the n-r most significant address bits define a partition.
Recirculating cube	Any set of n-r PE address bits defines a partition.
Recirculating Illiac	For N > 2, the Illiac network cannot be partitioned.
Recirculating PM2I	The n-r least significant PE address bits define a partitioning.

VI. THE AUGMENTED DATA MANIPULATOR NETWORK

VI.1. Introduction

The Augmented Data Manipulator network (ADM) is a multistage PM2I network (Figure III.7) where each switching cell of the network receives control signals independently of any other cell of the network. This chapter explores some of the capabilities of the ADM. Section 2 compares the types of data transfers that the ADM can accomplish with those which can be done by the generalized cube network. Section 3 illustrates how some data transfers can be accomplished by more than one control setting. Also considered here are some transfers which the ADM cannot perform in one pass through the network. Section 4 presents some group theoretic properties of the ADM network.

VI.2. The ADM and the Generalized Cube Network.

The data manipulator network (Figure III.7) consists of n stages with N switching cells per stage. The stages are ordered from n-1 to 0, where the interconnection functions of stage i are $PM2_{+i}$, $PM2_{-i}$, and the identity (straight across). The controls of the data manipulator are limited to one pair per stage. Cells whose i th address bit is 0 respond to one control; cells whose i th bit is 1 respond to the other control. Because groups of N/2 cells respond to one control signal, the data manipulator network cannot simulate the generalized cube network with individual interchange box control. Since the hardware cost of the data manipulator is greater than that of the generalized cube with four function interchange boxes (Chapter IV), there are many cases when the data manipulator is not a cost effective option.

The ADM is a data manipulator with individual cell control, i.e., each cell can receive none, one, two, or three of the signals H (straight across), U ($PM2_{-i}$), and D ($PM2_{+i}$) [SIS78]. The data output from cell P and stage i becomes the dat input to cell P' at stage i-1, where $P' \in \{P, (P+2^i) \text{ modulo } N, (P-2^i) \text{ modulo } N\}$. Each cell passes data independently of any other cell. The ADM can perform all the interconnection functions of the generalized cube network with four function boxes.

<u>Theorem VI.1</u>: The ADM can simulate the generalized cube network with individual box control [SIS78].

<u>Proof</u>: Consider simulating the four function box at stage i of the generalized cube whose upper input is x and lower input is y. Let the

subscript x denote the ADM control for cell x at stage i. Then, the four functions of the interchange box can be simulated as follows.

straight - $H_x H_y$,

exchange - $D_x D_y$,

lower broadcast - $U_y H_y$,

upper broadcast - $D_x H_x$. []

Recall from Chapter II that data transfers from one PE to another can be represented as a mapping from the set of input PE addresses to the set of output PE addresses. When this mapping is one-to-one and onto, the data transfer will be called a <u>permutation of input PE addresses to output PE addresses</u>. For a given network, if a control setting exists which accomplishes permutation f(P) of input PE addresses to output PE addresses, then the network <u>passes the permutation f(P)</u>.

A permutation, f(P), from PE addresses to PE addresses can be written as:

$$\begin{pmatrix} 0 & 1 & 2 & \cdots & N-1 \\ f(0) & f(1) & f(2) & \cdots & f(N-1) \end{pmatrix}$$

where the top line is the input PE address and the bottom line is the output PE address to which f maps the input. A permutation can cyclicly permute a set of elements $i_0, i_1, \ldots, i_r$ such that

$$f(i_0) = i_1,\ f(i_1) = i_2,\ \ldots\ ,\ f(i_r) = i_0.$$

Write this cycle as $(i_0\ i_1\ \ldots\ i_r)$. The physical interpretation of this cycle is that input i_0 sends its data to output i_1, input i_1 sends its data to output i_2, ..., input i_{r-1} sends its data to output i_r, and input i_r sends its data to output i_0. Any permutation can be written as a product of disjoint cycles. For example, if $N = 8$ and $f(P)$ is the permutation

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 5 & 4 & 3 & 7 & 1 & 6 & 0 \end{pmatrix}$$

Then, $f(P)$ can be written as the product of four cycles:

$$(0\ 2\ 4\ 7)(1\ 5)(3)(6).$$

That is, 0 sends data to 2, 2 sends data to 4, 4 sends data to 7, 7 sends data to 0, 1 and 5 exchange data, and 3 and 6 pass straight across the network.

The permutations of input PE addresses to output PE addresses which the ADM network can accomplish are a superset of those the generalized cube network of Figure III.6 can perform [SIS78]. In this section, some classes of permutations which the ADM can perform, but the generalized cube cannot, are enumerated.

Theorem VI.2: For $N \geq 4$, consider an SIMD machine with an ADM network that is partitioned under multiple control partitioning. Let the PEs be divided into 2^{n-2} groups of $2^2 = 4$ PEs such that the addresses of the PEs in each group are

$$Q * 2^{n-2} + P;$$
$$0 \leq Q < 4, \text{ for some fixed } P;$$
$$0 \leq P < 2^{n-2}.$$

Then, within each group of four PEs, the ADM can form all 24 permutations of four items.

Proof: Let a control C_P^i affect cell P at stage i. If C = H, then the connection is straight across. If C = U, then data is moved from cell P at stage i to cell $PM2_{-i}(P)$ at stage i-1. If C = D, then data moves from cell P at stage i to cell $PM2_{+i}$ (P) at stage i-1. Table VI.1 lists the 24 possible permutations of 4 PEs along with the ADM controls with produce these permutations for a 2-stage ADM network. □

__Theorem VI.3:__ The generalized cube network cannot perform all permutations of four PEs.

__Proof:__ Let the addresses of the four PEs be 0, 1, 2, and 3. Then eight permutations cannot be passed by a 2-stage generalized cube network. These eight are expressed as cyclic permutations, as follows.

(0) (1 2) (3)

(0) (1 3 2)

(0 1 2) (3)

(0 1 3 2)

(0 2 3 1)

(0 2 3) (1)

(0 3 1) (2)

(0 3) (1) (2).

For example, the permutation (0) (1 2) (3) cannot be performed by the generalized cube network. For N = 4, the data from PE(0) and from PE(3) must pass straight across both stages 1 and 0 of the 2-stage network. So, the controls for the two interchange boxes which affect PE(0) must be set to no exchange, and the controls for the two interchange boxes which affect PE(3) must be set to no exchange. If PE(1) and PE(2) exchange data, then the path that the data from PE(1) takes should be from PE(001) to position 011 at the output of stage 1 to PE(010) at the output of stage 0. Since data from PE(3) travels straight across the

Table VI.1: For N=4, the ADM forms of all 24 possible permutations of 4 PE addresses.

PERMUTATION	CONTROLS	
	STAGE 1	STAGE 0
(0) (1) (2) (3)	$H_0^1H_1^1H_2^1H_3^1$,	$H_0^0H_1^0H_2^0H_3^0$
(0) (1) (2 3)	$H_0^1H_1^1H_2^1H_3^1$,	$H_0^0H_1^0D_2^0U_3^0$
(0) (1 2) (3)	$H_0^1H_1^1H_2^1H_3^1$,	$H_0^0D_1^0U_2^0H_3^0$
(0 1) (2) (3)	$H_0^1H_1^1H_2^1H_3^1$,	$D_0^0U_1^0H_2^0H_3^0$
(0 2) (1) (3)	$D_0^1H_1^1U_2^1H_3^1$,	$H_0^0H_1^0H_2^0H_3^0$
(0 3) (1) (2)	$H_0^1H_1^1H_2^1H_3^1$,	$U_0^0H_1^0H_2^0D_3^0$
(0) (2) (1 3)	$H_0^1D_1^1H_2^1U_3^1$,	$H_0^0H_1^0H_2^0H_3^0$
(0 1) (2 3)	$H_0^1H_1^1H_2^2H_3^1$,	$D_0^0U_1^0D_2^0U_3^0$
(0 3) (1 2)	$H_0^1H_1^1H_2^1H_3^1$,	$U_0^0D_1^0U_2^0D_3^0$
(0 2) (1 3)	$D_0^1D_1^1U_2^1U_3^1$,	$H_0^0H_1^0H_2^0H_3^0$
(0 1 2) (3)	$D_0^1H_1^1U_2^1H_3^1$,	$H_0^0D_1^0U_2^0H_3^0$
(0 2 3) (1)	$D_0^1H_1^1U_2^1H_3^1$,	$U_0^0H_1^0H_2^0D_3^0$

Table VI.1. (Continued)

(0 3 2) (1)	$D_0^1 H_1^1 U_2^1 H_3^1$,	$H_0^0 H_1^0 D_2^0 U_3^0$
(0 2 1) (3)	$D_0^1 H_1^1 U_2^1 H_3^1$,	$D_0^0 U_1^0 H_2^0 H_3^0$
(0 1 3) (2)	$H_0^1 D_1^1 H_2^1 U_3^1$,	$D_0^0 U_1^0 H_2^0 H_3^0$
(0 3 1) (2)	$H_0^1 D_1^1 H_2^1 U_3^1$,	$U_0^0 H_1^0 H_2^0 D_3^0$
(0) (1 2 3)	$H_0^1 D_1^1 H_2^1 U_3^1$,	$H_0^0 H_1^0 D_2^0 U_3^0$
(0) (1 3 2)	$H_0^1 D_1^1 H_2^1 U_3^1$,	$H_0^0 D_1^0 U_2^0 H_3^0$
(0 3 2 1)	$H_0^1 H_1^1 H_2^1 H_3^1$,	$U_0^0 U_1^0 U_2^0 U_3^0$
(0 3 1 2)	$D_0^1 D_1^1 U_2^1 U_3^1$,	$H_0^0 H_1^0 D_2^0 U_3^0$
(0 2 3 1)	$D_0^1 D_1^1 U_2^1 U_3^1$,	$U_0^0 H_1^0 H_2^0 D_3^0$
(0 2 1 3)	$D_0^1 D_1^1 U_2^1 U_3^1$,	$D_0^0 U_1^0 H_2^0 H_3^0$
(0 1 3 2)	$D_0^1 D_1^1 U_2^1 U_3^1$,	$H_0^0 D_1^0 U_2^0 H_3^0$
(0 1 2 3)	$H_0^0 H_1^0 H_2^0 H_3^0$,	$D_0^0 D_1^0 D_2^0 D_3^0$

network, a conflict occurs between PE(1) and PE(3) at stage 1. []

Theorems VI.2 and VI.3 indicate that even for small values of N, the ADM is a more complex network than the generalized cube. Some classes of permutations which can be performed by the ADM but not by the generalized cube can be defined.

<u>Theorem VI.4</u>: Let P and Q be two PEs such that, for some i, $0 \leq i < n-1$, $Q = P \pm 2^i$, and the addresses of P and Q differ in two or more bit positions. Then, the ADM can pass the permutation

$$(0)\ (1)\ \ldots\ (P\ Q)\ \ldots\ (N-1).$$

The generalized cube cannot pass this permutation.

<u>Proof</u>: For $Q = P + 2^i$, set all stages except stage i to "straight." At stage i, P employs $PM2_{+i}$ and Q employs $PM2_{-i}$. That is, P and Q exchange data at stage i, and so no conflicts can occur in passing this permutation. For example, if N = 16, i = 4, P = 0100, and Q = 1000, then $PM2_{+4}(P) = PM2_{-4}(Q)$.

For $Q = P - 2^i$, set all stages except stage i to "straight." At stage i, P employs $PM2_{-i}$ and Q employs $PM2_{+i}$. So, P and Q exchange data at stage i, and no conflicts result in passing the data through the network.

The generalized cube cannot perform this type of permutation. P and Q differ in two or more bit positions, and so more that one stage of the network must execute a cube function. If this happened, then data from PEs other than P and Q would be affected. []

In Chapter IV, section IV.3 showed that a multistage Shuffle-Exchange network cannot perform a perfect shuffle (Figure II.6a) in one pass through the network. Since the generalized cube network is topologically equivalent to a multistage Shuffle-Exchange network, the generalized cube cannot perform a perfect shuffle in one pass through the network. However, the ADM can.

<u>Theorem VI.5</u>: The ADM can perform a perfect shuffle in one pass through the network.

<u>Proof</u>: The control settings for stage i, $n > i \geq 0$, are determined as follows. The address of a cell P at stage i is $P = p_{n-1} \cdots p_1 p_0$.

```
set stage n-1 to straight across;
for i = n-2 step -1 until 0
    if $p_{i+1} \neq p_i$ then
            if $p_{i+1} = 0$ then set cell P at stage i to $PM2_{+i}$;
                else if $p_{i+1} = 1$
                    then set cell P at stage i to $PM2_{-i}$;
            else set stage i to straight across;
```

For the controls calculated from the algorithm, after stage i, data originally from PE $P = p_{n-1} \cdots p_1 p_0$ is at the output of cell $P' = p_{n-2} p_{n-3} \cdots p_i p_{n-1} p_{i-1} \cdots p_0$. So, after passing through the ADM network using the settings calculated above, data from input P is moved to output PE(P') where $P' = p_{n-2} \cdots p_0 p_{n-1}$. Figure VI.1 shows the data movement through an ADM for N=8 for the perfect shuffle permutation.

At stage i of the network, data from input cell $P = p_{n-2} p_{n-3} \cdots p_{i+1} p_{n-1} p_i p_{i-1} \cdots p_0$ is moved by the algorithms to output cell

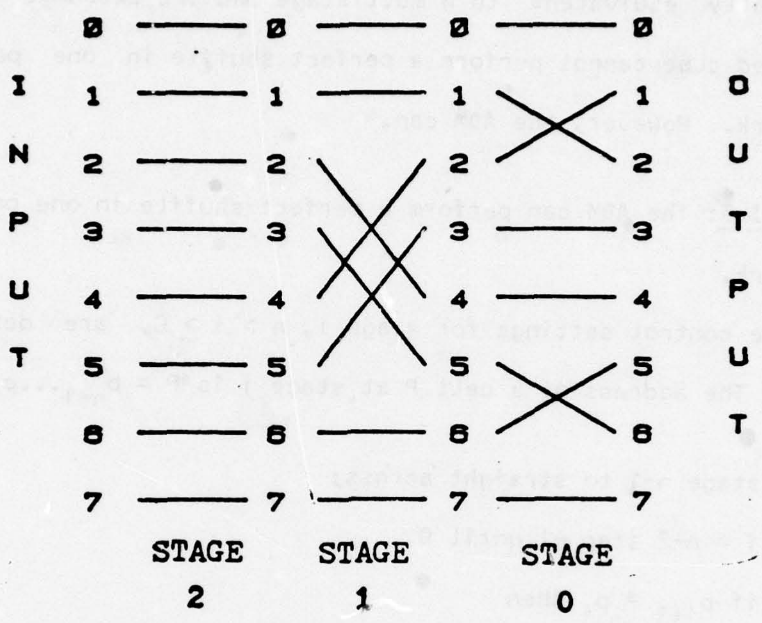


Figure VI.1: For N = 8, the ADM can perform the perfect shuffle permutation.

$P' = p_{n-2}\cdots p_{i+1}p_i p_{n-1}p_{i-1}\cdots p_0$. For all cells that have addresses where $p_i = p_{n-1}$, the "straight across" connection is used. Thus, $P = P'$ and no conflicts of data can occur for these cells of the network. If $p_i \neq p_{n-1}$, then two cases are considered.

(1) $P = p_{n-2}\cdots p_{i+1}01p_{i-1}\cdots p_0$ moves data to output cell $P' = p_{n-2}\cdots p_{i+1}10p_{i-1}\cdots p_0$. The only input cell position at stage i that could conflict with the data from P is cell $Q = p_{n-2}\cdots p_{i+1}10p_{i-1}\cdots p_0 = P'$. But, the algorithm moves this data to output cell P at the input to stage i-1. So, data from P and data from Q cannot conflict at stage i.

(2) $P = p_{n-2}\cdots p_{i+1}10p_{i-1}\cdots p_0$ is moved by the algorithm to output cell $P' = p_{n-1}\cdots p_{i+1}01p_{i-1}\cdots p_0$. This case is analogous to case (1). There is no input cell position of the network that the algorithm allows to conflict with P.

Since no data can conflict with any other data when it moves in a straight across path, when it moves in a $PM2_{+i}$, or when it moves in a $PM2_{-i}$ path, the algorithm generates control settings which allow conflict free passage of the data through the network. []

VI.3. Capabilities and Limitations of the ADM

For some data transfers, more than one control setting may exist for the ADM network. This is not true for the generalized cube network, where stage i and only stage i can change bit p_i as the input address is mapped to the output address.

For example, consider the number of legal control settings of the ADM, those for which no conflicts occur between data passing through the network. As an illustration, consider an ADM network for N = 8 (Figure III.7). Stage 2 performs $PM2_{\pm 2}$. Since $PM2_{+2} = PM2_{-2}$, stage 2 can perform $Cube_2$. That is, data can be conditionally exchanged by cell positions whose addresses differ in bit p_2. There are four such pairs of cells at stage 2. So, the number of legal control settings is $2^{8/2} = 16$.

Stage 1 can perform $PM2_{\pm 1}$. From Corollary V.6, two independent groups of PEs pass data among themselves at stage 1. These are the even numbered PEs, 0, 2, 4, and 6, and the odd numbered PEs, 1, 3, 5, and 7. Since the two groups are independent, the number of legal control settings for this stage is square of the number for one of the groups. Consider the group of even numbered PEs. There are nine legal control settings. If the four PEs are grouped to conditionally exchange data between PEs $p_2 00$ and PEs $p_2 10$ (Figure VI.2), as in the cube network, then $2^2 = 4$ legal control settings exist. If the PEs are grouped to conditionally exchange data between PEs $p_2 00$ and PEs $\overline{p_2} 10$ (Figure VI.3), then $2^2 = 4$ legal control settings exist. Both of these groups contain the identity (straight across) setting, and so the two groupings exhibit $2*2^2 - 1 = 7$ distinct legal control settings. One setting

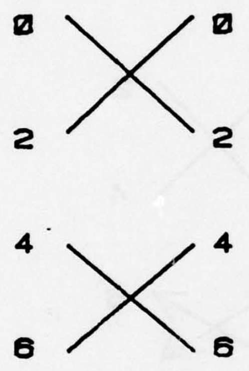


Figure VI.2: Stage 1 of the ADM for N = 8 can conditionally exchange data between PEs P and (P+2) modulo N for P = X00.

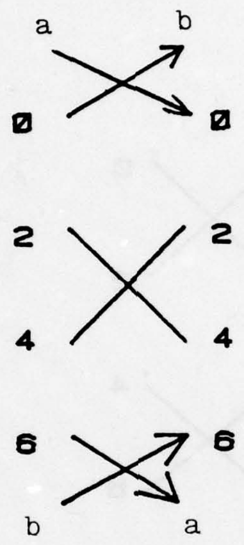


Figure VI.3: Stage 1 of the ADM can conditionally exchange data between PEs P and (P+2) modulo N for P = X10.

exists which maps P to P', where P' = (P + 2) modulo N, and one setting exists which P to P', where P' = (P − 2) modulo N. These 7 + 1 + 1 = 9 control settings are the only legal settings for the four even PEs at stage 1. Any other settings cause conflicts to occur. So, the total number of legal control settings at stage 1 is $9^2 = 81$.

The number of legal control settings for stage 0 can be similarly counted. Data is moved among all 8 PEs at this stage. Data can be conditionally exchanged between PEs P and P+1 for P even to yield 2^4 = 16 legal control settings. Data can be conditionally exchanged between PEs P and P+1, for P odd, to yield 2^4 = 16 control setting. These two cases give 2*16 − 1 = 31 distinct legal control settings. The final two cases move data from PE(P) to PE(P') for P' = (P + 1) modulo N, and from PE(P) to PE(P') for P' = (P − 1) modulo N. The number of distinct legal control settings for stage 0 is 31 + 1 + 1 = 33.

The total number of legal control settings for a 3-stage ADM network is (16)*(81)*(33) = 42,768. But, the number of permutations of eight items is 8! = 40,320. So, there must be some permutations that can be performed using more than one control settings; some control settings must be redundant. The next theorem demonstrates one form of these redundant settings.

<u>Theorem</u> <u>VI</u>.<u>6</u>: Consider the interconnection function f(x) = Cube_i, for all i, $0 \le i < n-1$, for all x, $0 \le x < N$. Then, there are n−i different control settings for the ADM which realize f(x).

<u>Proof:</u> Cube_i exchanges data between $P = p_{n-1}\cdots p_{i+1}0p_{i-1}\cdots p_0$ and $P' = p_{n-1}\cdots p_{i+1}1p_{i-1}\cdots p_0$. So, $P = (P' - 2^i)$ modulo N, and $P' = (P + 2^i)$ modulo N. Cube_i can be realized by setting the ADM controls such that

at stage i, cells whose i^{th} address bit equal 0 perform $PM2_{+i}$, while those whose i^{th} address bit is 1 perform $PM2_{-i}$. Since, $2^i = 2^{i+1} - 2^i$, an equivalent control setting is for PEs P to execute $PM2_{+(i+1)}$ and $PM2_{-i}$, and for PEs P' to execute $PM2_{-(i+1)}$ and $PM2_{+i}$. Since,

$$2^i = 2^{i+1} - 2^i$$
$$= 2^{i+2} - 2^{i+1} - 2^i$$
$$= 2^k - 2^{k-1} - \ldots - 2^i, n < k < i,$$

there are n-i different settings for the ADM which accomplish $Cube_i$.

Since all PEs with $p_i = 0$ execute $PM2_{+j}$ ($PM2_{-j}$), while PEs with $p_i = 1$ execute $PM2_{-j}$ ($PM2_{+j}$), no collisions occur in passing data through the network. []

Consider the uniform shift permutations that send data from PE(P) to PE(P') where $P' = (P + A)$ modulo N, $0 < A < N$, for all PEs. The generalized cube network can form this permutation of PE addresses in one pass through the network, due to [LAW75] and the topological equivalence of the multistage Shuffle-Exchange network and the generalized cube [SIS78]. However, only one distinct control setting exists for a given uniform shift permutation. The ADM has redundant settings to realize this permutation. Let the binary representation of A be $a_{n-1} \ldots a_1 a_0$.

<u>Theorem VI.7</u>: The ADM has redundant control settings for all uniform shifts of A, $0 < A < N$.

<u>Proof:</u>

Case 1: $0 \leq A < N/2$. If $a_i \in \{0, 1\}$, then the ADM can be set according to the following.

At stage i, if $a_i = 0$, then set the network to straight across. If $a_i = 1$, then set the network to $PM2_{+i}$.

Let A be expressed in signed digit notation, where $a'_i \in \{0, +1, -1\}$. So, A is expressed as the sum and difference of powers of 2^i. For example, for $N = 8$, $A = 0111$ can also be expressed as $A = 100(-1) = 8 - 1$, as $A = 10(-1)1 = 8 - 2 + 1$, and as $A = 1(-1)11 = 8 - 4 + 2 + 1$. Thus, the following are all equivalent representations of A, for $N \geq 8$, where "1...11" indicates a string of one or more 1's.

$$a'_{n-1} \ldots a'_k 01 \ldots 110 a'_j \ldots a'_0$$
$$a'_{n-1} \ldots a'_k 10 \ldots 0(-1)0 a'_j \ldots a'_0$$
$$a'_{n-1} \ldots a'_k 10 \ldots (-1)10 a'_j \ldots a'_0$$
$$a'_{n-1} \ldots a'_k 1(-1)1 \ldots 10 a'_j \ldots a'_0$$

where $n < k < j \leq 0$. Each of these different representations of A can be used to yield control settings for the ADM network according to to following algorithm

At stage i, if $a_i = 0$, then set stage i to straight across. If $a_i = 1$, then set stage i to $PM2_{+i}$. If $a_i = -1$, then set stage i to $PM2_{-i}$.

Case 2: $N/2 \leq A < N$. The corresponding uniform shift permutation is the same as the permutation $f(P) = (P-B)$ modulo N, for

$0 \leq B < N/2$. For this case, there are settings of the network equivalent to that of case 1. For example, for $N = 16$, $B = 0011 = 010(-1) = 1(-1)0(-1)$. Thus, case 2 can be proven in a manner analogous to case 1. []

If the stages of the ADM network are traversed in reverse order, that is, the interconnection functions of the first stage are $PM2_{\pm(0)}$, the interconnection functions of the i^{th} stage are $PM2_{\pm(i)}$, and the interconnection functions of the last stage are $PM2_{\pm(n-1)}$, the resulting network is called the <u>Inverse Augmented Data Manipulator</u> (IADM). Figure VI.4 shows the IADM for $N = 8$. Stage i of both the ADM and IADM performs $PM2_{\pm i}$, but the first stage of the ADM is labeled stage n-1 while the first stage of the IADM is labeled stage 0.

<u>Lemma VI.1</u>: The ADM passes the permutation f if any only if the IADM passes the inverse permutation, f^{-1}.

<u>Proof</u>: Suppose the ADM performs the mapping from P to

$$f(P) = (P + I_{n-1}2^{n-1} + I_{n-1}2^{n-2} + \ldots + I_0 2^0) \text{ modulo } N,$$

$$I_j \in \{0, +1, -1\},\ 0 \leq j < n, \text{ for all } P,\ 0 \leq P < N.$$

Each input PE(P) has its own set of I_j to specify its path through the network. At stage j, if $I_j = 0$ for a given cell, then that cell sends data straight across. If $I_j = +1$, then cell Q sends data to $Q + 2^j$. If $I_j = -1$, then cell Q sends data to $Q - 2^j$.

Recall that the first stage of the IADM is labelled stage 0, while the first stage of the ADM is labelled stage n-1. Set the stages of the IADM as follows. If input cell Q at stage i of the ADM is set to $PM2_{+i}$ ($PM2_{-i}$), then set input cell $Q+2^i$ at stage i of the IADM to $PM2_{-i}$

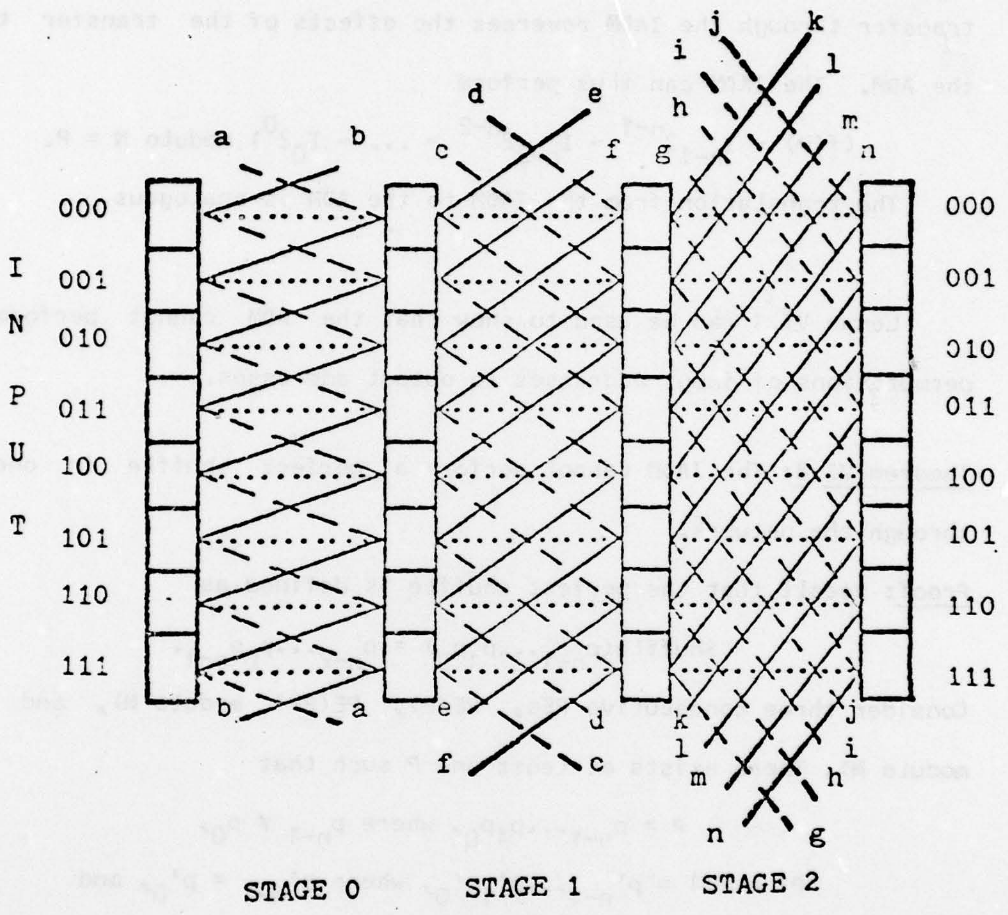


Figure VI.4: The Inverse ADM (IADM) is a multistage PM2I network where the first stage conditionally performs $PM2_{\pm 0}$, the second, $PM2_{\pm 1}$, and so forth, and the last stage $PM2_{\pm(n-1)}$.

($PM2_{+i}$). In traversing the ADM, if data passes from input cell Q to output cell $Q+2^i$, then in traversing the IADM, the data passes from input cell $Q+2^i$ to output cell Q. If cell Q at stage i of the ADM is set to straight, then so is cell Q of stage i of the IADM. The data transfer through the IADM reverses the effects of the transfer through the ADM. The IADM can thus perform

$$(f(P) - I_{n-1}2^{n-1} - I_{n-2}2^{n-2} - \ldots - I_0 2^0) \text{ modulo } N = P.$$

The translation from the IADM to the ADM is analogous. []

Lemma VI.1 can be used to show that the ADM cannot perform some permutations of input addresses to output addresses.

Theorem VI.8: The IADM cannot perform a perfect shuffle in one pass through the network.

Proof: Recall that the perfect shuffle is defined as

$$\text{Shuffle}(p_{n-1}\ldots p_1 p_0) = p_{n-2}\ldots p_0 p_{n-1}.$$

Consider three consecutive PEs, PE(P), PE(P+1 modulo N), and PE(P-1 modulo N). There exists at least one P such that

$$P = p_{n-1}\ldots p_1 p_0, \text{ where } p_{n-1} \neq p_0,$$

$$P' = P+1 = p'_{n-1}\ldots p'_1 p'_0, \text{ where } p'_{n-1} = p'_0, \text{ and}$$

$$P'' = P-1 = p''_{n-1}\ldots p''_1 p''_0, \text{ where } p''_{n-1} = p''_0.$$

For example, for N = 16, one such set is P = 0011, P+1 = 0100, and P-1 = 0010. In general, one such P is $0^{n-2}11$. The difference of the addresses P and Shuffle(P), | P - Shuffle(P)|, is an odd number if and only if $p_{n-1} \neq p_0$. Since no combination of $PM2_{+i}$ and $PM2_{-i}$, $0 < i < n$, yields an odd number, data from P must use $PM2_{\underline{+}0}$ at stage 0. But, data from P' = P+1 modulo N and from P'' = P-1 modulo N must use the straight across connection at stage 0. Since $PM2_{+0}(P) = (P + 1)$ modulo $N = P'$, and $PM2_{-0}(P) = (P - 1)$ modulo $N = P''$, a conflict occurs at stage 0 between P and P' or between P and P''. Therefore, the IADM cannot perform a perfect shuffle in one pass through the network. []

Corollary VI.1: The ADM cannot perform an inverse perfect shuffle in one pass through the network.

Proof: Follows from Lemma VI.1 and Theorem VI.8. []

A bit reversal algorithm transfers data from PE $P = p_{n-1}\ldots p_1p_0$ to PE $P' = p_0p_1\ldots p_{n-1}$. Data is sent to the PE whose address is the reverse of the current location. This permutation is useful in the computation of fast fourier transforms [SS79].

Theorem VI.9: For $N > 8$, the IADM cannot perform a bit reversal in one pass through the network.

Proof: Assume that all PEs whose addresses in binary are palindromes, i.e., they are the same read from the left or from the right, such as 00100 and 01010, do not pass data. This frees these positions in the network to be used by the remaining positions. An argument based on eliminating the palindromes will give the most optimistic results.

Consider three consecutive binary PE addresses, P, P + 1, and P - 1 such that none of the three are palindromes. $P = p_{n-1} \ldots p_1p_0$, where $p_{n-1} \neq p_0$, $P + 1 = p'_{n-1} \ldots p'_1p'_0$, where $p'_{n-1} = p'_0$, and $P - 1 = p''_{n-1} \ldots p''_1p''_0$, where $p''_{n-1} = p''_0$. For example, P could be $0^{n-1}1$.

A conflict occurs as follows. The distance between P and its bit reversal, Pr, $|P - Pr|$, is an odd number. Since no combination of $PM2_{+i}$ and $PM2_{-i}$, $0 < i < n$, yields an odd number, $PM2_{+0}$ or $PM2_{-0}$ must be applied to P. Thus, $loc_1 (P) = P + 1$ or $P - 1$. $P + 1$ and $P - 1$ are an even distance from the positions of their bit reversals. Thus, neither $PM2_{+0}$ or $PM2_{-0}$ can be applied to data from $P + 1$ or $P - 1$. So, $loc_1 (P + 1) = P + 1$, and $loc_1 (P - 1) = P - 1$. At stage 0, a conflict occurs between P and $P + 1$ or between P and $P - 1$.

For $N = 8$, beginning with address 000, for each three consecutive addresses, at least one is a palindrome. The conflict situation above cannot occur. []

<u>Corollary VI.2</u>: The ADM cannot perform a bit reversal in one pass through the network.

<u>Proof</u>: Follows from Lemma VI.1 and Theorem VI.9. []

VI.4. Some Theoretical Results

Consider all mappings from the input addresses to the output addresses that are bijections (one-to-one and onto), i.e., the set of all permutations of N items. The ADM cannot perform all N! such permutations in a single pass through the network. The ADM can perform some subset of all permutations in one pass through the network called the allowable permutations of the ADM. Consider applying two or more mappings to data passing through the ADM, that is, making multiple passes through the network, where each pass realizes one of the allowable permutations of the ADM. This is called composition of mappings.

Theorem VI.10: The set of all allowable permutations of the ADM which can be performed in one pass through the network is not a group under composition of mappings.

Proof: Let S be the set of all allowable permutations of the ADM. The perfect shuffle permutation is a member of S, but neither of its possible inverses, the inverse shuffle or (n-1) perfect shuffles can be realized by the ADM in one pass through the network. So, since S does not contain the inverse of all its members, S is not a group. []

Consider the set defined by the mappings of Theorem VI.4 and all compositions of these mappings. These permutations exchange one or more pairs of PEs, (P_k, Q_k), such that for any pair k, $Q_k = P_k \pm 2^i$, $0 \leq i < n$, $0 \leq P_k, Q_k < N$, and P_k and Q_k differ in two or more bit positions. Add to this set the identity mapping, and call the resulting set of permutations T.

<u>Theorem VI.11</u>: The set T for the ADM is a group under composition of mappings.

<u>Proof</u>: Any member of T exchanges a pair of PEs by applying only one interconnection function, either $PM2_{+i}$ or $PM2_{-i}$, to that pair. The inverse mapping is attained by applying $PM2_{-i}$ or $PM2_{+i}$ to these same two PEs. So, this inverse mapping is in T.

Let a, b, and c be three mappings in T. Because these mappings affect pairs of PEs, the order of their application to the data is irrelevant, that is,

$$(a * b) * c = a * (b * c),$$

where * is composition of mappings. So, T is assciative.

Finally, the identity permutation is in T, and so T is a group. []

VI.5. Conclusions

This chapter has pointed out some of the capabilities and limitations of the ADM. The set of mappings of input PE addresses to output PE addresses which the ADM can perform is a superset of those of the generalized cube network. Some classes of permutations have been presented which can be performed by the ADM but not by the generalized cube network. The ADM also has multiple control settings for some mappings, such as the Cube_i functions and the uniform shift functions. If one path through the network is not operating, then an alternate route can be selected. So, the network has fault tolerant properties for some permutations. The ADM network cannot perform all N! permutations of N input PE addresses to output PE addresses. This chapter has shown two such permutations, the inverse shuffle and the bit reversal permutations.

VII. PARALLEL ALGORITHMS AND INTERCONNECTION NETWORKS

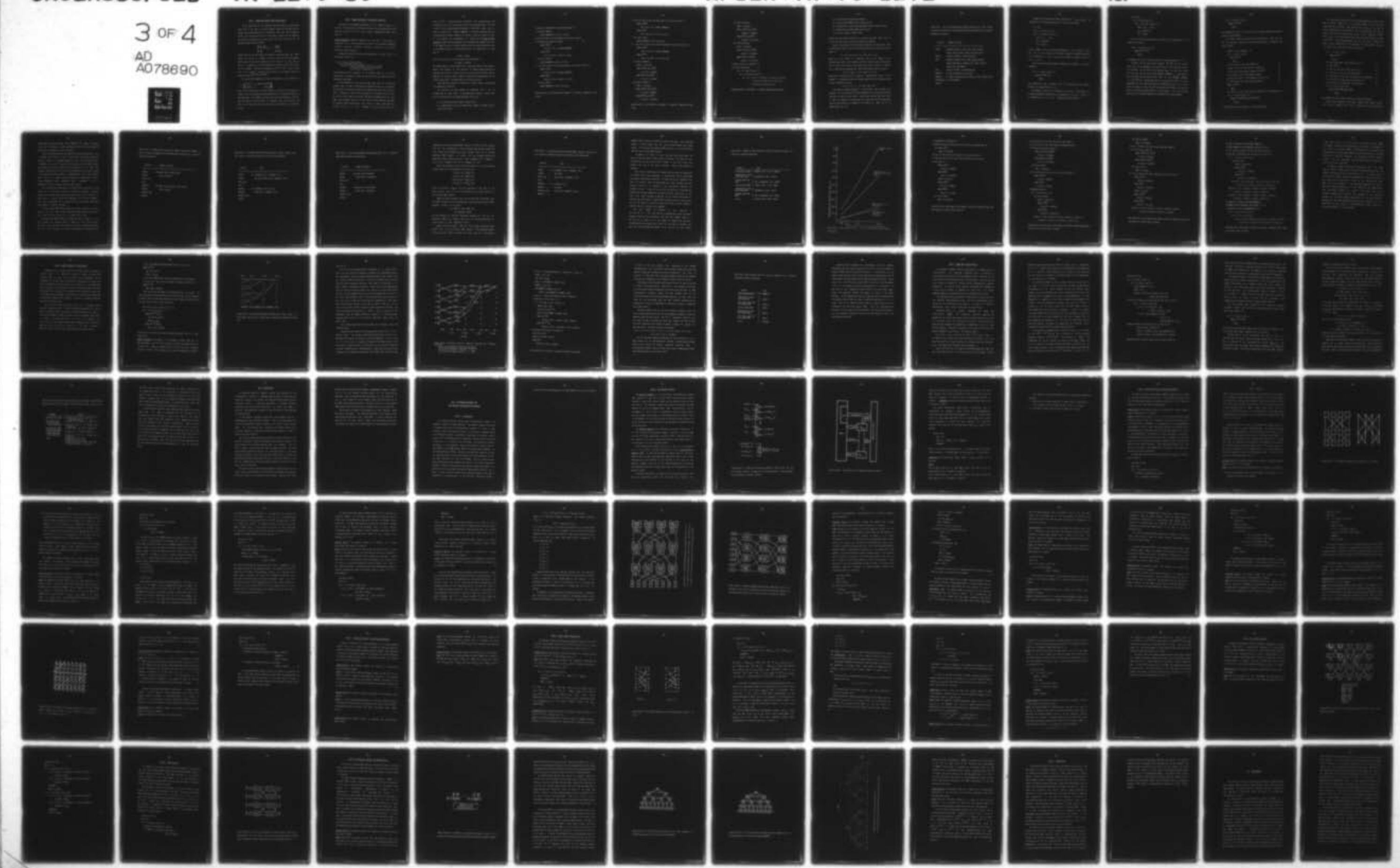
VII.1. Introduction

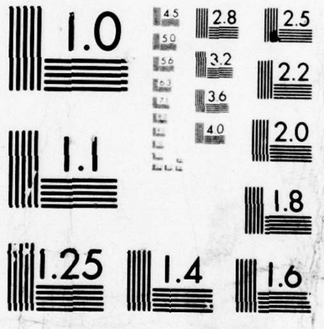
In designing algorithms for parallel computers, some researchers assume that no data transter time penalties are incurred in the computation [SAM77, HEL77, KST73]. Other investigators have indicated that this may not be a reasonable assumption [GEN78]. This chapter investigates the impact of the interconnection network on the performance of selected parallel algorithms for image processing.

Image processing tasks are well suited to parallel processing [BS78a, CDL78, SIE78], and many parallel computer architectures have been proposed for the task [FU78]. Yet, the effects of the interconnection network on the efficiency of an algorithm have not been explored. This chapter selects several representative image processing tasks and formulates a parallel algorithm for each. For each algorithm, the time required to pass data among PEs is examined for various interconnection networks. Section 2 defines the picture that will be used for all algorithms of this chapter. It describes how the picture is distributed among the N PEs of the parallel computer system. Section 3 presents a parallel smoothing algorithm. Section 4 considers formation of a histogram of the grey levels of the picture. This histogram can then be used to threshold the picture at a specific grey level in order to extract subpictures of the image. Section 5 presents a classification algorithm which assigns a pixel to one of M classes based on certain statistics known about each class.

AD-A078 690 PURDUE UNIV LAFAYETTE IN SCHOOL OF ELECTRICAL ENGINEERING F/G 9/2
DESIGN AND ANALYSIS OF INTERCONNECTION NETWORKS FOR PARTITIONAB--ETC(U)
AUG 79 S D SMITH , H J SIEGEL AFOSR-78-3581
UNCLASSIFIED TR-EE79-39 AFOSR-TR-79-1272 NI

3 OF 4
AD
A078690





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

VII.2. Image Processing: Data Distribution

In the algorithms for this chapter, the data image is a black-and-white 512 X 512 pixel (picture element) image with 128 grey levels. Suppose that 1024 processors are available, and that each processor stores a 16 X 16 block of the 512 X 512 image. Assume that the 1024 PEs are logically arranged as an array of 32 X 32 PEs, and that the PE addresses range from 0 to 1023:

```
PE 0    PE 1  . . .  PE 31
PE 32   PE 33 . . .  PE 63
              . . .
PE 992        . . .  PE 1023
```

Assume that the 16 X 16 blocks are stored in row major order. So, PE(0) stores the pixels of columns 0 to 15 of rows 0 to 15 of the images, PE(1) stores the pixels of columns 16 to 31 of rows 0 to 15, and so forth. In general, $N = 2^n$ PEs operate on a picture of $2^k \times 2^k$ pixels, for k an integer. The PEs are arranged in a $2^{n/2} \times 2^{n/2}$ array, where n/2 is an integer, such that each PE stores in its memory a block of pixels of size $2^{k-(n/2)} \times 2^{k-(n/2)}$.

For notational purposes, let each PE consider its 16 X 16 matrix as

$$H = \begin{bmatrix} h(0,0) & \ldots & h(0,14) & h(0,15) \\ & \ldots & & \\ h(15,0) & \ldots & & h(15,15) \end{bmatrix}.$$

Also, let the subscripts of h(i,j) extend to -1 and 16, if necessary, in order to aid in calculations across boundaries of two adjacent blocks in different PEs. For example, the pixel to the left of h(0,0) is h(0,-1), and the pixel below h(15,15) is h(16,15). So, $-1 \le i,j \le 16$. Let $h(i, j \rightarrow k)$ refer to the set of pixels h(i,j), h(i,j+1), ..., h(i,k-1), h(i,k).

VII.3. Image Processing: A Smoothing Algorithm

One goal of any smoothing algorithm is to suppress noise in a picture. The algorithm of this section does this by averaging the noise over all 8 pixels in a 3 X 3 pixel square neighborhood about pixel h(i,j).

Problem statement: Smooth an image, h, of size 512 X 512 pixels by replacing h(i,j) by the average grey level of its 8 nearest neighbors, h(i,j+1), h(i,j-1), h(i-1,j), h(i+1,j), h(i-1,j-1), h(i-1,j+1), h(i+1,j+1), and h(i+1,j-1).

A general algorithm to perform the smoothing on pixel h(i,j) to yield pixel hs(i,j) is:

```
for i = 0 step +1 until 15 do
    for j = 0 step +1 until 15 do
        hs(i,j) = 1/8 * ( h(i+1,j) + h(i-1,j) + h(i,j+1)
                  + h(i,j-1) + h(i+1,j-1) + h(i+1,j+1)
                  + h(i-1,j+1) + h(i-1,j-1) ).
```

The approach to this algorithm is to perform 1024 16 X 16 pixel evaluations in parallel, rather than one 512 X 512 evaluation as in the sequential algorithm.

At the boundaries of the 16 X 16 array, data must be transmitted between PEs in order to calculate the smoothed value, hs. For example, h(-1,0) must be transferred from the PE 'above' the local PE, except for PEs 0 through 31, those at the 'top edge' of the logical array of PEs. If a PE is at the edge of the array, and has no neighboring PE at that edge, the boundary values of h will be set to the value at the edge. For example, PE(0) contains the first block of data, and does not receive h(-1, 0 -> 15). Instead, it sets h(-1, 0 -> 15) to h(0, 0 -> 15). To take these data transfers into consideration, the following steps must be executed before the algorithm above. "Set ICN to PE+j" sets the interconnection network so that PE(P) sends data to PE(P + j modulo N). "BLOCK TRANSFER := vector" indicates that the interconnection network transfers the vector of data as a block, as fast as the hardware can accommodate it. Recall that the data transfer registers (DTRs) pass data between the interconnection network and a PE. A PE loads data into its DTRin register, and the network moves the data to the DTRout register of the destination PE. The instruction to load data is

DTRin := data;

and the instruction to retrieve data from the network is

variable := DTRout;

PE address masks are used extensively in the algorithms of this chapter. Recall from Chapter II that positive PE address masks specify the addresses of all PEs to remain active for the instructions which follow. Negative PE address masks specify which PEs are deactivated for the coming instructions. No mask implies all PEs are active.

Algorithm VII.1 passes all the data in blocks before undertaking the smoothing calculation.

The transfers of data needed for Algorithm VII.1 can be accomplished by many kinds of interconnection networks. Seven types will be considered for the algorithm above. They are:

a) a recirculating Cube network (Figure IV.5),

b) a combinational logic multistage Cube network (Figures III.3, III.5, and III.6),

```
** Move all data before calculations begin. **
 Set ICN to PE+32;
     BLOCK TRANSFER := h(15, 0 -> 15);
 ** Turn off PEs on the top edge of the array of PEs **
 ** and input all data as a block.                      **
     MASK [-0^5X^5]
          h(-1, 0 -> 15) := BLOCK TRANSFER;
     MASK [0^5X^5]
          h(-1, 0 -> 15) := h(0, 0 -> 15);
 Set ICN to PE-32;
     BLOCK TRANSFER := h(0, 0 -> 15);
 ** Turn off all PEs on the bottom edge of the array of PEs. **
     MASK [-1^5X^5]
          h(16, 0 -> 15) := BLOCK TRANSFER;
     MASK [1^5X^5]
          h(16, 0 -> 15) := h(15, 0 -> 15);
 Set ICN to PE+1;
     BLOCK TRANSFER := h(0 -> 15, 15);
```

Algorithm VII.1 (continued on next page): A parallel smoothing algorithm.

```
** Turn off PEs on the left hand edge of the array of PEs. **
   MASK [-$X^5 0^5$]
      h(0 -> 15, -1) := BLOCK TRANSFER;
   MASK [$X^5 0^5$]
      h(0 -> 15, -1) := h(0 -> 15, 0);
 Set ICN to PE-1;
   BLOCK TRANSFER := h(0 -> 15, 0);
** Turn off all PEs on the right hand edge of the array of PEs. **
   MASK [-$X^5 1^5$]
      h(0 -> 15, 16) := BLOCK TRANSFER;
   MASK [$X^5 1^5$]
      h(0 -> 15, 16) := h(0 -> 15, 15);
 Set ICN to PE+32+1;
   DTRin := h(15,15);
   MASK [-$0^5 X^5$] OR [-$X^5 0^5$]
      h(-1,-1) := DTRout;
   MASK [$0^5 X^5$] OR [$X^5 0^5$]
      h(-1,-1) := h(0,0);
 Set ICN to PE+32-1;
   DTRin := h(15,0);
   MASK [-$0^5 X^5$] OR [-$X^5 1^5$]
      h(-1,16) := DTRout;
   MASK [$0^5 X^5$] OR [$X^5 1^5$]
      h(-1,16) := h(0,15);
```

Algorithm VII.1 (continued on next page): A parallel smoothing algorithm.

```
  Set ICN to PE-32-1;
      DTRin := h(0,0);
      MASK $[-1^5X^5]$ OR $[-X^51^5]$
           h(16,16) := DTRout;
      MASK $[1^5X^5]$ OR $[X^51^5]$
           h(16,16) := h(15,15);
  Set ICN to PE-32+1;
      DTRin := h(0,15);
      MASK $[-1^5X^5]$ OR $[-X^50^5]$
           h(16,-1) := DTRout;
      MASK $[1^5X^5]$ OR $[X^50^5]$
           h(16,-1) := h(15,0);
** Data transfer is complete. **
** Do the calculation.        **
      for i = 0 step +1 until 15
         for j = 0 step +1 until 15
             hs (i,j) := 1/8 * ( h(i+1,j) + h(i-1,j) + h(i,j+1)
                        + h(i,j-1) + h(i+1,j-1) + h(i+1,j+1)
                        + h(i-1,j+1) + h(i-1,j-1) );
```

Algorithm VII.1 (continued): A parallel smoothing algorithm.

c) a pipelined multistage Cube network,

d) a recirculating PM2I network (Figure IV.6),

e) a combinational logic multistage PM2I network (Figure III.7),

f) a pipelined multistage PM2I network, and

g) an Illiac network (Figure II.8).

Assume that the network available is as wide as the data word, and so can pass one data word in one pass through the network.

Recall from Chapter IV the following formulas for calculating data transfer delay times. For a recirculating network, the time to pass one datum is

$$Trc = dr + dm + drn + dr = dm + drn + 2\ dr,$$

where dr is the delay of a register, drn is the delay of the combinational logic of the network, and dm is the delay of a multiplexer used to select data input to the network from either DTRin or DTRout. An $n = \log_2 N$ stage combinational logic multistage network has delay

$$Tm = dr + dms * n + dr = dms * n + 2\ dr,$$

where dms is the delay of one stage of combinational logic of the network. A k-stage pipelined multistage network has the following delay to pass S segments of data:

$$Tk = dr + (k + S - 1)*(n/k * dms + dr).$$

The number of passes through a recirculating Cube network that executes the interconnection function needed for the parallel smoothing algorithm is given by Table VII.1. Recall that sending data from PE(P) to PE(P + 2^i modulo N) is equivalent to saying that the interconnection function maps address P to address P + 2^i modulo N. That is, 2^i is added, modulo N, to P.

Table VII.1: The recirculating Cube network requires the given number of passes to execute the interconnection functions needed for a parallel smoothing algorithm.

function	number of passes
PE+32	maximum 5 passes for all PEs, Cube_5 through Cube_9, times 16 data items: total 80 passes.
PE-32	similar to PE+32: total 80 passes.
PE+1	maximum 5 passes for all PEs, Cube_0 through Cube_4 (some PEs are masked off), times 16 data items: total 80 passes.
PE-1	similar to PE+1: total 80 passes.
PE+32+1, PE+32-1, PE-32-1,and PE-32+1	at most 10 passes for all PEs, for each interconnection function, times 4 data items: total 40 passes.

Assume all PEs know their binary PE address $P = p_{n-1} \cdots p_1 p_0$. An algorithm for the binary addition of 2^i to P is as follows.

```
$p_i := \overline{p_i}$;
if $p_i := 0$ then Cin := 1;
for j = i+1 step +1 until n-1
    Cout := $p_j \cdot$ Cin;
    $p_j := p_j \oplus$ Cin;
    Cin := Cout;
```

A one is added in bit p_i, thus complementing p_i. Any resultant carry bit, Cin or Cout, equal to one ripples through the remaining address bits, changing $p_j = 1$ to $p_j = 0$, until the ripple is stopped by some $p_j = 0$, $i < j < n$.

The recirculating Cube network simulates this addition as follows [SIE77b].

```
for j = i step +1 until n-1
    MASK [$X^{n-j}0^{j-i}X^i$]
        Cube$_j$;
```

The command "Set ICN to PE+32" with a recirculating Cube network executes this algorithm for i = 5.

The mapping from P to P - 2^i modulo N is similar. The borrow, Bin or Bout, propagates beginning at p_i, changing $p_j = 0$ to $p_j = 1$, until it is stopped by $p_j = 1$, $i < j < n$. A subtraction algorithm is

```
$p_i := \overline{p}_i$;
if $p_i = 1$ then Bin := 1;
    for j = i+1 step +1 until n-1
        Bout := $\overline{p}_j \cdot$ Bin;
        $p_j := p_j \oplus$ Bin ;
        Bin := Bout;
```

The recirculating Cube network simulates this subtraction of 2^i modulo N as follows.

```
for j = i step +1 until n-1
    MASK [$X^{n-j}1^{j-i}X^i$]
        Cube$_j$;
```

The command "Set ICN to PE-32" executes the algorithm above for i = 5.

Consider the "Set ICN to PE+1" command. PEs $X^5 0^5$ do not receive data, and so are masked off by Algorithm VII.1 after the data transfer is initiated. So, PEs $X^5 1^5$ do not transfer data. For PE+1, the active PEs that must receive data are PEs $X^5$00001, $X^5$00010, ... , and $X^5$11111. So, the active PEs that must execute PE+1 are PEs $X^5$00000, $X^5$00001, ... , and $X^5$11110. The maximum number of bit positions that a carry bit propagates for this addition occurs for PE($X^5$01111), which is a carry chain of length 5. Thus, the PE+1 interconnection function is simulated by the recirculating Cube network in five passes. The recirculating Cube uses the following algorithm to accomplish the data movement.

```
for i = 0 step +1 until 4
    MASK [$X^{10-i}0^i$]
        Cube$_i$;
```

The algorithm for PE-1 is similar, where the maximum length borrow chain occurs for PE($X^5$10000).

The "PE+33" connection combines the algorithms for P+32 and P+1. The recirculating Cube uses the following algorithm to accomplish "Set ICN to PE+33."

```
** Add 1 to all addresses.  **
for j = 0 step +1 until 4
    MASK [$X^{n-j}0^j$]
        Cube$_j$;
** At this point, only PEs ($X^5 0^5$) will                      **
** have a carry out from the first addition.                     **
** They hold the data which began in PEs $X^5 1^5$.              **
** For them, $p_5$ will not change, because                      **
** 1(the carry out) + 1(the addition of 32) + $p_5$ =            **
** 1(the carry out of $p_5$), $p_5$(the sum bit).                **
MASK [$-X^{n-5}0^5$]
    Cube$_5$;
**  Add the carries resulting from adding 32 to the addresses. **
for j = 6 step +1 until n-1
    MASK [$X^{n-j}0^{j-5}X^5$] OR [$X^{n-j}0^{j-6}X0^5$]
        Cube$_j$;
```

The PE-33 connection executes a similar algorithm.

The recirculating Cube uses the following algorithm to execute "Set ICN to PE+31." Adding 31 to an address P adds 1 in all five least significant bit positions. Thus, address bits p_4, p_3, p_2, p_1, and p_0 are all complemented by the addition. The only PE addresses that do not have a carry out of bit position p_4 are the initial addresses $X^{n-5}0^5$. After step five of the addition algorithm, data from these PEs resides in PE $X^{n-5}1^5$. Since these addresses now have the data intended for them, they are masked off for the remainder of the algorithm. The last five steps of the algorithm continue an addition.

```
for i = 0 step +1 until 4
    MASK [$X^{n-i}1^i$]
        Cube$_i$;
**  Data that began in PEs ($X^5 0^5$) are now in               **
**  PEs ($X^5 1^5$).                                              **
**  For only this set of PEs, there is no                        **
**  carry out of bit $p_4$.  So, they do not                      **
**  participate in the last five steps of the algorithm.        **
**  The last five steps add only the carry out                   **
** of bit $p_4$ to the remainder of the address.                  **
for j = 5 step +1 until n-1
    MASK [$X^{n-j}0^{j-5}X^5$] AND [$-X^5 1^5$]
            Cube$_j$;
```

The command "Set ICN to PE-31" executes a similar algorithm.

A total time of 160(dm + drn + 2*dr) + 160(dm + drn + 2*dr) + 40(dm + drn + 2*dr) = 360(dm + drn + 2*dr) is required for the recirculating

Cube network to pass the data. From [SIE79b], the number of passes shown here represents a lower time bound for the Cube interconnection functions to accomplish the data transfers.

A multistage Cube network can execute any shift permutation, i.e., a mapping from PE(x) to PE(x + j modulo N), in one pass through the network. Theorem 1 of [LAW73] showed this result for the omega network. In [PEA77], this result is shown for the indirect binary n-cube network. The network can form any uniform shift permutation. It is shown that repeated applications of the unit shift permutation can be done instead in one pass through the network. A combinational logic Cube network would require the number of passes indicated in Table VII.2. A total time of 32(dms*10 + 2*dr) + 32(dms*10 + 2*dr) + 4(dms*10 + 2 dr) = 68(dms*10 + 2*dr) is needed to pass the data items.

Let the network be pipelined into 5 pieces, where each piece is two stages of the multistage network followed by a register. A 5-stage pipelined multistage Cube network would require the time indicated in Table VII.3. A total time of 4(dr +20(dms*2 + dr)) + 4(dr + 5(dms*2 + dr)) = 8*dr + 100(dms*2 + dr) is required for the pipelined multistage Cube network to transfer the data.

A recirculating PM2I network passes the data according to Table VII.4. The total time for the recirculating PM2I network to pass the data is (64 + 8)*(dm + drn + 2 dr) = 72(dm + drn + 2 dr).

The multistage PM2I network can execute any PE+j transfer in one pass through the network, -1024 < j < 1024, i.e., PE 1, PE 31, PE 32, and PE 33. Thus, any one of the transfers for this algorithm requires one use of the network to reach its destination. The time for the

Table VII.2: A combinational logic cube network requires the number of passes listed to accomplish the interconnection functions for a parallel smoothing algorithm.

function	number of passes
PE+32,	one pass times 16 data items:
PE-32,	total 64 passes.
PE+1, and	
PE-1	
PE+32+1,	one pass for each times 4 data items:
PE+32-1,	total 4 passes.
PE-32+1, and	
PE-32-1	

Table VII.3: A 5-stage pipelined multistage cube network passes data for a parallel smoothing algorithm in the times shown above.

function	time
PE+32, PE-32, PE+1, and PE-1.	dr + 5(dms*2 + dr) + 15(dms*2 + dr) for each: total 4(dr + 20(dms*2 + dr)).
PE+32+1, PE+32-1, PE-32+1, and PE-32-1	dr + 5(dms*2 + dr) for each: total 4(dr + 5(dms*2 + dr))

Table VII.4: A recirculating PM2I network passes data for a parallel smoothing algorithm as shown above.

function	number of passes
PE+32, PE-32, PE+1, and PE-1	one pass for each datum: total 16(4) = 64 passes
PE+32+1, PE+32-1, PE-32+1, and PE-32-1	two passes for each datum: total 4(2) = 8 passes.

combinational logic multistage PM2I network to transfer the data equals that for the combinational logic multistage cube network to pass the data. Table VII.5 indicates the data transfer times for a 5-stage pipelined PM2I network. To transfer all the boundary values, the pipelined network requires time Tk = 4(dr + 5(dms*2 + dr) + 15(dms*2 + dr)) + 4(dr + 5(dms*2 + dr)) = 8 dr + 100(dms * 2 + dr).

Recall from Chapter II that the Illiac network is a recirculating network, where the interconnection functions are

$$\mathrm{Illiac}_{+1}(P) = \mathrm{PM2}_{+0}(P)$$

$$\mathrm{Illiac}_{-1}(P) = \mathrm{PM2}_{-0}(P)$$

$$\mathrm{Illiac}_{+m}(P) = \mathrm{PM2}_{+n/2}(P)$$

$$\mathrm{Illiac}_{-m}(P) = \mathrm{PM2}_{-n/2}(P)$$

where $m = \sqrt{N}$ and $n = \log_2 N$. For this algorithm, m = 32, and so the data transfer time for the Illiac network is the same as that for a recirculating PM2I network.

Compare the data transfer times for the pipelined multistage Cube and PM2I networks and the combinational logic multistage Cube and PM2I networks. If dms = dr = D, then:

$$Tk = 4(61D) + 64D = 308D, \text{ and}$$

$$Tm = 68(12D) = 816D.$$

For this example, the percent difference between the time for the pipelined network to transfer data and the unpipelined network to transfer data is ((816 - 308)/816) = 62%.

Compare the data transfer time for the 5-stage pipelined PM2I network and the recirculating PM2I network. The pipelined network requires time Tk = 308 D to transfer the data, and the recirculating

Table VII.5: A 5-stage pipelined multistage PM2I network passes data for a parallel smoothing algorithm according to the table above.

function	time
PE+32,	dr + 5(dms*2 + dr) + 15(dms*2 + dr)
PE-32,	for each:
PE+1, and	total 4(dr + 20(dms*2 + dr)).
PE-1.	
PE+32+1,	dr + 5(dms*2 + dr)
PE+32-1,	for each:
PE-32+1, and	total 4(dr + 5(dms*2 + dr)).
PE-32-1.	

network requires time Trc = 288 D to transfer the data. The pipelined network is 6.9% slower than the recirculating network, and is more costly. In this case, the pipelined PM2I network is less cost effective than the recirculating PM2I network.

In summary, if D = dms = dr = drn = dm, then Table VII.6 shows the delay to pass the data for each network considered. The times will vary according to the size of the picture stored in each PE, since this affects the size of any block transfer. Figure VII.1 shows this variance.

The results of Algorithm VII.1 showed that when data is transferred in blocks, some time advantage is gained when using a pipelined multistage interconnection network. An algorithm can be designed that does not transfer data in large blocks, but rather transfers data one word at a time as it is required in the calculation. Consider the boundary points. First, hs for row 0 of the matrix H is calculated by transfering the data needed for one point, and then calculating hs for that pixel. Next, the same is done for row 15, column 0 and column 15. Finally, the calculation is completed by determining hs for those points not at the edge of H, that is, those that require no data transfers. Algorithm VII.2 illustrates this approach.

The total number of transfers for this algorithm is 22 + 22 + 12 + 12 = 68. The time for a combinational logic multistage network to execute these transfers is Tm = 68 (Sm (10*dms + 2*dr)) = 68 (12 Sm D), where Sm is the number of segments per data word transmitted by the network, and D = dms = dr. The time to transfer the data for a k-stage pipelined network is Tk = 68 (dr + (k + Sk - 1)(n/k

Table VII.6: Summary of data transfer times for different networks for a parallel smoothing algorithm.

network	time
recirculating Cube	360(dm + drn + 2 dr) = 1440 D
combinational logic multistage Cube	68(dms*10 + 2*dr) = 816 D
5-stage pipelined Cube	8dr + 100(dms*2 + dr) = 308 D
recirculating PM2I	72(dm + drn + 2 dr) = 288 D
combinational logic multistage PM2I	68(dms*10 + 2*dr) = 816 D
5-stage pipelined PM2I	8dr + 100(dms*2 + dr) = 308 D
Illiac	72(dm + drn + 2 dr) = 288 D

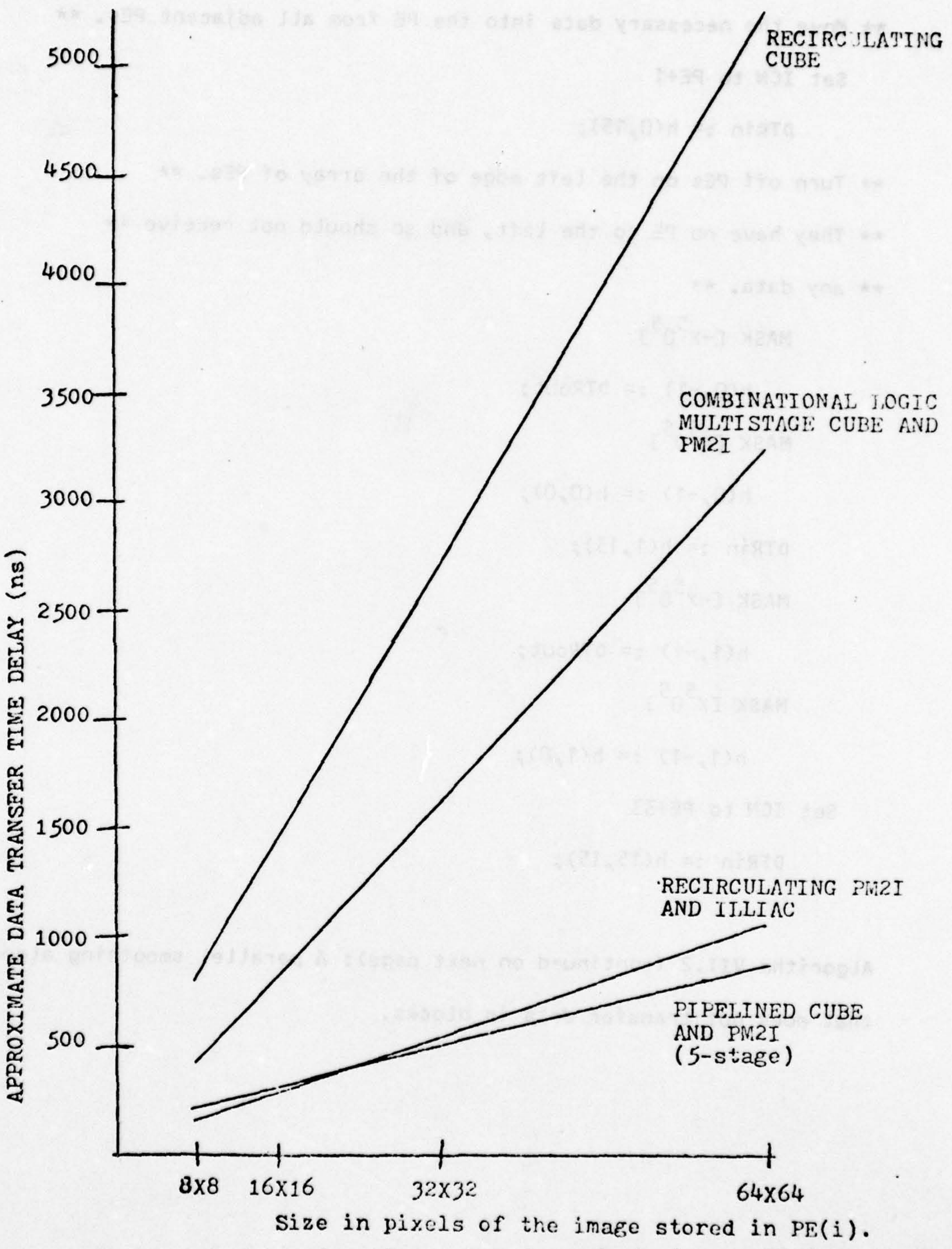


Figure VII.1: Data transfer times vs number of pixels per PE for Algorithm VII.1.

```
** CALCULATE hs FOR ROW 0 **
** Move the necessary data into the PE from all adjacent PEs. **
   Set ICN to PE+1
       DTRin := h(0,15);
** Turn off PEs on the left edge of the array of PEs. **
** They have no PE to the left, and so should not receive **
** any data. **
       MASK [-X$^5$0$^5$]
          h(0,-1) := DTRout;
       MASK [X$^5$0$^5$]
          h(0,-1) := h(0,0);
       DTRin := h(1,15);
       MASK [-X$^5$0$^5$]
          h(1,-1) := DTRout;
       MASK [X$^5$0$^5$]
          h(1,-1) := h(1,0);
   Set ICN to PE+33
       DTRin := h(15,15);
```

Algorithm VII.2 (continued on next page): A parallel smoothing algorithm that does not transfer data in blocks.

```
** Turn off PEs on the top row and the left edge **
** of the array of PEs. They have no neighboring PEs **
** from which to receive data. **
      MASK [-$0^5X^5$] OR [-$X^5 0^5$]
         h(-1,-1) := DTRout;
      MASK [$0^5X^5$] OR [$X^5 0^5$]
         h(-1,-1) := h(0,0);
   Set ICN to PE+32
      DTRin := h(15,0);
** Turn off PEs on the top row of the array of PEs. **
      MASK [-$0^5X^5$]
         h(-1,0) := DTRout;
      MASK [$0^5X^5$]
         h(-1,0) := h(0,0);
** CALCULATE ROW 0 **
   for j = 0 to 14 by +1 do
      Set ICN to PE+32
         DTRin := h(15,j+1);
         MASK [-$0^5X^5$]
            h(-1,j+1) := DTRout;
         MASK [$0^5X^5$]
            h(-1,j+1) := h(0,j+1);
      hs(0,j) := 1/8 * (h(1,j) + h(-1,j) + h(0,j+1) + h(0,j-1)
         + h(1,j-1) + h(1,j+1) + h(-1,j+1) + h(-1,j-1) );
```

Algorithm VII.2 (continued on next page): A parallel smoothing algorithm that does not transfer data in blocks.

```
    Set ICN to PE+32-1
        DTRin := h(15,0);
 ** Turn off PEs in the top row and right hand edge **
 ** of the array of PEs. **
        MASK [-$0^5X^5$] OR [-$X^51^5$]
            h(-1,16) := DTRout;
        MASK [$0^5X^5$] OR [$X^51^5$]
            h(-1,16) := h(0,15);
    Set ICN to PE-1;
        DTRin := h(0,0);
 ** Turn off PEs on the right hand edge of array of PEs. **
        MASK [-$X^51^5$]
            h(0,16) := DTRout;
        MASK [$X^51^1$)
            h(0,16) := h(0,15);
        DTRin := h(1,0);
        MASK [-$X^51^5$]
            h(1,16) := DTRout;
        MASK [$X^51^5$]
            h(1,16) := h(1,15);
    hs(0,15) := 1/8 * (h(1,15) + h(-1,15) + h(0,16) + h(0,14)
                 + h(1,14) + h(1,16) + h(-1,16) + h(-1,14))
```

Algorithm VII.2 (continued on next page): A parallel smoothing algorithm that does not transfer data in blocks.

```
** ROW 15 IS DONE IN AN ANALOGOUS MANNER **

** TOTAL NUMBER OF DATA TRANSFERS FOR ROW 0 = 22 **

** For column 0, assume that h(14,-1) and h(15,-1) **

** have been transferred for calculation for row 15 **

    for i = 1 step +1 until 12 do

        Set ICN to PE+1

            DTRin := h(i+1,15);
            MASK [-0^5 X^5]
                h(i+1,-1) := DTRout;
            MASK [0^5 X^5]
                h(i+1,-1) := h(i+1,0);
        hs(i,0) := 1/8 * (h(i+1,0) + h(i-1,0) + h(i,1) + h(i,-1)
                   + h(i+1,-1) + h(i+1,1) + h(i-1,1) + h(i-1,-1))
    hs(13,0) := 1/8 * (h(14,0) + h(12,0) + h(13,1) + h(13,-1)
                + h(14,-1) + h(14,1) + h(12,1) + h(12,-1) )
    hs(14,0) := 1/8 * (h(15,0) + h(13,0) + h(14,1) + h(14,-1)
                + h(15,-1) + h(15,1) + h(13,1) + h(13,-1))

** COLUMN 15 IS DONE IN THE SAME MANNER **

** NUMBER OF DATA TRANSFERS FOR COLUMN 0 = 12 **

** CALCULATE hs FOR THE REMAINDER OF H **

    for i = 1 step +1 until 14 do

        for j = 1 to 14 by +1 do

            hs(i,j) := 1/8 * (h(i+1,j) + h(i-1,j) + h(i,j+1) + h(i,j-1)

                + h(i+1,j-1) + h(i+1,j+1) + h(i-1,j+1) + h(i-1,j-1) )
```

Algorithm VII.2 (continued): A parallel smoothing algorithm that does not transfer data in blocks.

* dms + dr)) = 68 D (1 + (k + Sk - 1)(n/k + 1)), where Sk is the number of segments per data word transmitted by the network. For example, if Sk = 1 and k = 5, then Tk = 1088 D. The times for the recirculating and combinational logic multistage networks to do the transfers are the same as the first algorithm presented. The pipelined version, however, differs, since the advantages seen by block transfer of data are not found in the second algorithm.

In summary, a parallel algorithm for smoothing a 512 X 512 image requires $0(16^2) = 0(256)$ steps to process the image plus any extra time needed to transfer data to the required PEs. If it is assumes that a recirculating network requires one computational time unit to pass one data item, then the data transfer times for the recirculating cube, PM2I, and Illiac networks is not negligible compared to the time to perform the smoothing operation. For the pipelined and combinational logic multistage networks, the transfer of 68 data items is not negligible compared to the time for the remainder of the algorithm, although the pipelined network algorithm will fare better than the combinational logic multistage network using block transfers. For this algorithm, the best choice of interconnection network would be a recirculating PM2I or an Illiac network.

VII.4. Image Processing: Thresholding

Segmentation of a picture extracts the subpictures of the image for further study. If a subpicture contains a range of grey levels different from the rest of the picture (the background), then a histogram of the grey level distribution of the pixels will show a peak which corresponds to the grey levels of the subpicture. The subpictures are extracted from the image by selecting threshold values of the grey levels which are the values at the bottoms of the valleys on either side of the peak of the histogram. This method is called threshold detection by mode method [ROK76, CDL77, CDL78]. A parallel histogram formation algorithm is defined below.

The calculations for this task are based on a technique called recursive doubling [ST75]. A recursive doubling algorithm is a type of parallel algorithm which, using N PEs, combines N items in $O(\log_2 N)$ steps. As an example, consider forming the sum of N numbers, D(0), D(1), ... , D(N-1). Let PE(i) store D(i), $0 \leq i < N$, in its memory. The first step of the algorithm forms N/2 intermediate sums in parallel. Each sum is D(i) + D(i-1), for i even, $0 \leq i < N$, and is stored in PE(i). For the first step of the algorithm, then, the PEs execute

```
** All odd numbered PEs pass DATA, D(i), for i odd. **
MASK [$X^{n-1}1$]
    SET ICN to PE-1;
    DTRin := DATA;
** All even numbered PEs add the incoming data to the DATA, **
** D(i), for i even, that is stored in the memory of PE(i). **
MASK [$X^{n-1}0$]
    SUM = DATA + DTRout;
```

After each step, the number of intermediate sums is halved. The last step sums two intermediate sums to give the final sum of N numbers. A general recursive doubling algorithm to sum N numbers using N PEs with the final sum in PE(0) is described below.

```
sum := D(i), for all PEs i, $0 \leq i < N$.
for i = 0 step +1 until n - 1
    MASK [$X^{n-(i+1)}10^i$]
        set ICN to PE-($2^i$);
        DTRin := sum;
    MASK [$X^{n-(i+1)}00^i$]
        sum := sum + DTRout;
```

Figure VII.2 illustrates the recursive doubling process for N = 8 items using 8 PEs.

Problem statement: The image h is distributed through 1024 PEs, as described above. Each PE has calculated a 128 bin histogram, HIST, for its piece of h. Merge the 1024 pieces of HIST into one histogram residing in PE(0). The histogram is now ready to determine a threshold

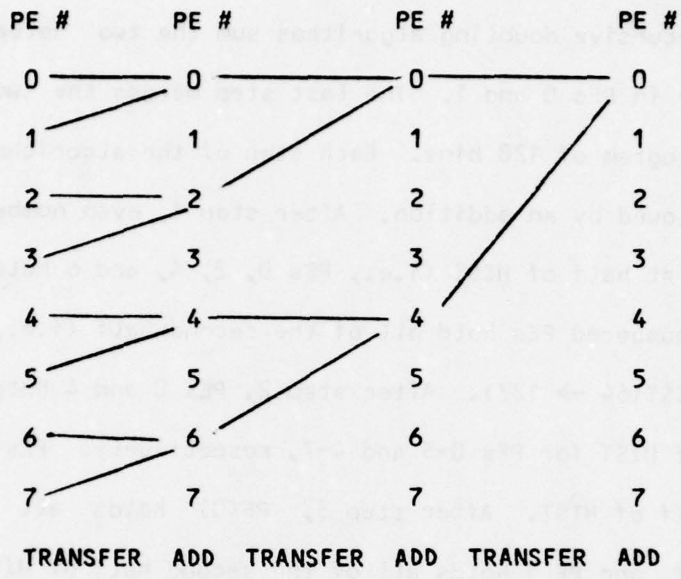


Figure VII.2: A recursive doubling algorithm sums 8 data items in 3 steps, where each step is a data transfer among PEs followed by an addition.

value for h.

For N = 8, the algorithm may be represented as in Figure VII.3. Until the last step, the histogram is passed as two independent halves. Two simultaneous recursive doubling algorithms sum the two halves, and the results reside in PEs 0 and 1. The last step merges the two halves into the final histogram of 128 bins. Each step of the algorithm is a data transfer followed by an addition. After step 1, even numbered PEs hold all of the first half of HIST (i.e., PEs 0, 2, 4, and 6 hold HIST(0 -> 63), and odd numbered PEs hold all of the second half (i.e., PEs 1, 3, 5, and 7 hold HIST(64 -> 127). After step 2, PEs 0 and 4 hold all of the first half of HIST for PEs 0-3 and 4-7, respectively. PEs 1 and 5 hold the second half of HIST. After step 3, PE(0) holds all of the first half of HIST, and PE 1 holds all of the second half of HIST. The final step merges the two halves in PE(0). Thus, N pieces of the histogram, HIST, are summed to PE(0) in ((log N) + 1) * (# bins)/2 data transfers and ((log N) + 1) * (# bins)/2 additions, where, here, (# bins) = 128.

For N = 1024, Algorithm VII.3 calculates the histogram using the technique above.

A sequential algorithm for calculating HIST requires 512 * 512 = 262,144 steps. The parallel algorithm described above uses 16 * 16 = 256 steps for each PE to calculate its local histogram. It uses $(\log_2 n + 1) * 128$ steps to merge the histogram into PE(0), where n = 10. Each step is a data transfer or an addition. So, the parallel algorithm uses 256 + 704(additions) + 704(transfers) = 1664 steps to give the same histogram as the sequential algorithm. This assumes that the time to add

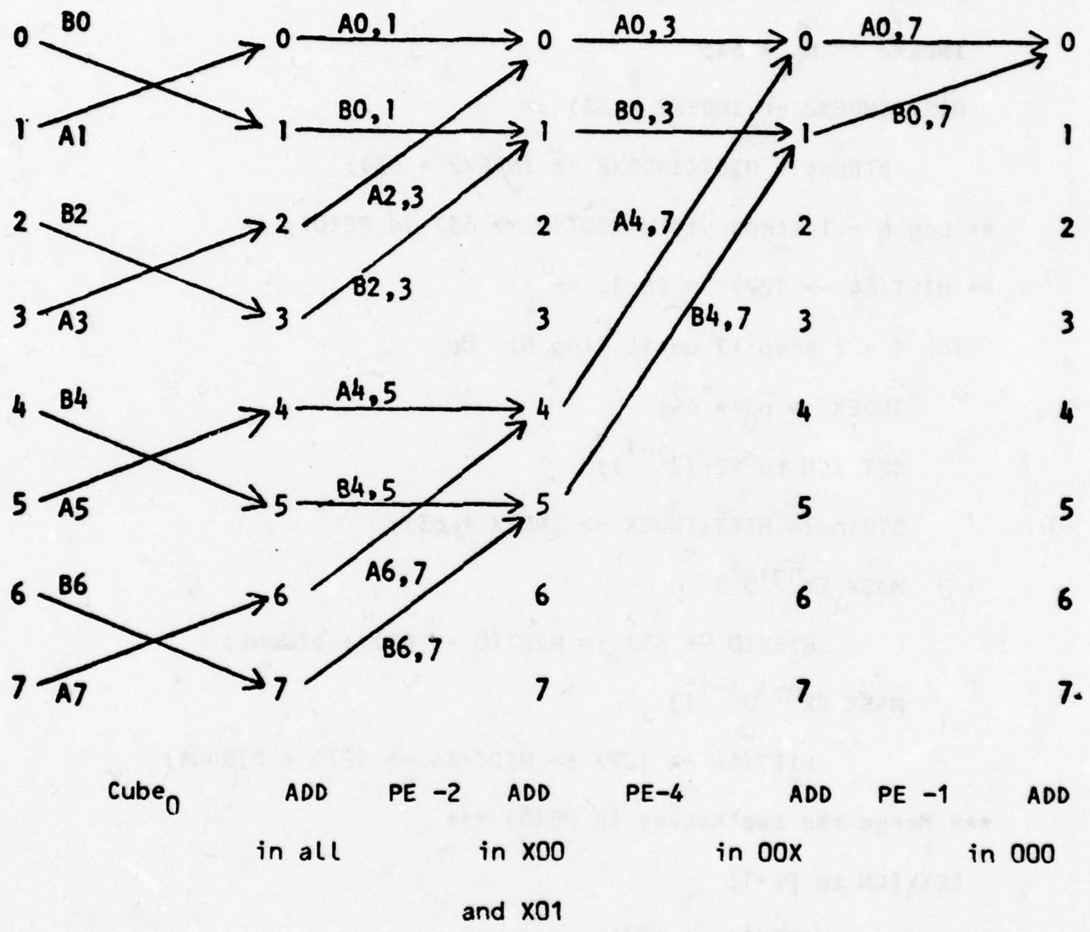


Figure VII.3: A modified recursive doubling algorithm for histogram formation.
Let Ai be the HIST(0 -> 63) data from PE(i).
Let Bi be the HIST(64 -> 127) data from PE(i).
Let Ai,j be the sum Ai + A(i+1) + ... + Aj.
Let Bi,j be the sum Bi + B(i+1) + ... + Bj.

```
** Step 1:  Exchange between PE i and PE i+1, i even. **
  INDEX := $\overline{p}_0$ * 64;
  SET ICN to Cube$_0$;
  DTRin := HIST(INDEX -> INDEX + 63);
  INDEX2 := $p_0$ * 64;
  HIST(INDEX2 -> INDEX2 + 63) :=
       DTRout + HIST(INDEX2 -> INDEX2 + 63);
** Log N - 1 steps yield HIST(0 -> 63) in PE(0), **
** HIST(64 -> 127) in PE 1. **
  for i = 2 step +1 until (log N)  do
      INDEX := $\overline{p}_0$ * 64;
      SET ICN to PE-($2^{i-1}$);
      DTRin := HIST(INDEX -> INDEX + 63);
      MASK [$X^{n-i}0^i$]
            HIST(0 -> 63) := HIST(0 -> 63) + DTRout;
      MASK [$X^{n-i}0^{i-1}1$]
            HIST(64 -> 127) := HIST(64 -> 127) + DTRout;
*** Merge the two halves in PE(0) ***
  SET ICN to PE-1;
  DTRin := HIST(64 -> 127);
  MASK [$0^n$]
      HIST(64 -> 127) := DTRout;
```

Algorithm VII.3: A parallel histogram formation algorithm.

is equal to the data transfer time. Depending on the network implementation, the calculation time difference between the serial and parallel algorithms is almost two orders of magnitude. Note that, in contrast to Algorithm VII.1, the data transfer times for the networks discussed are independent of the number of pixels stored in one PE.

Performing a data transfer time analysis similar to that of section 5 yields the results given in Table VII.7. The transfers of one step of the recursive doubling algorithm can be done using one Cube or PM2I interconnection function. Hence, the data transfer time for the recirculating Cube and recirculating PM2I networks are less than the times for the multistage Cube and PM2I networks. Because data is transferred in blocks, an advantage is realized by using a pipelined multistage network.

The data transfer time for the Illiac network indicates that the Illiac interconnection functions are not adequate to support a recursive doubling algorithm efficiently. The "PE -1" connection requires only one pass through the network per data item. However, the "PE -16" and "PE -512" connections each require 16 passes through the network for each data item. The total transfer time becomes

$$64\ (1 + 2 + 4 + 8 + 16 + 1 + 2 + 4 + 8 + 16 + 1)(dm + drn + 2\ dr)$$
$$= 64\ (63)\ (4D) = 16{,}128\ D.$$

For the histgram formation algorithm, the Illiac network is not a good choice for an interconnection network. A pipelined multistage network will afford the fastest algorithm execution time. A recirculating network is the next best choice, and a combinational logic multistage network the third best choice.

Table VII.7: Data transfer times for various networks for a parallel histogram formation algorithm.

network	time
recirculating Cube	2816 D
Combinational logic multistage Cube	8448 D
Five stage pipelined multistage Cube	2555 D
recirculating PM2I	2816 D
Combinational logic multistage PM2I	8448 D
Five stage pipelined multistage PM2I	2555 D
Illiac	16,128 D

Algorithm VII.3 performs two simultaneous recursive doubling algorithms, and after $\log_2 N$ steps half of the bins of the histogram are stored in PE(0) and half are stored in PE(1). A modification can be made to Algorithm VII.3 which further devides the histogram such that 2^b simultaneous recursive doubling algorithms take place, and after $\log_2 N$ steps, the first 2^b PEs each have $1/(2^b)$ of the bins of the histogram, $1 < b < n$. For example, an algorithm designed for $b = 2$ stores 1/4 of the bins of the histogram in each of PEs 0, 1, 2, and 3 after $\log_2 N$ steps. This algorithm reduces the number of items in each block transfer. If the histgram has 128 bins, Algorithm VIII.3 causes each PE to transfer 64 data items at a step. A modified algorithm for $b = 2$ causes each PE to transfer 32 data items at a step to give 1/4 of the histogram in four PEs after $\log_2 N$ steps. Since the interconnection functions are the same for both algorithms, the relative performance of all of the networks discussed is the same for both Algorithm VII.3 and the modified algorithm.

VII.5. Image Data Classification

One method of computer assisted interpretation of images relies on classification. The algorithm presented here is based on a classification method which is often applied to data collected by the Earth Resources Technology Satellite, called Landsat. Typically, ground cover classes of interest are defined having characteristic spectral properties. If there are M such classes, the probability of membership in one of the M classes is computed for each pixel. The highest computed probability determines the class assignment of the pixel.

The LARS algorithm was designed for data vectors X which consist of four spectral observations per pixel. Let X be a d-dimensional vector so that for this algorithm, h(i,j) refers to a vector of d spectral observations. Such a vector in PE(P) has the name $h^P(i,j)$.

Presented below is a parallel algorithm for image data classification [SW78]. The clustering algorithm supplies two parameters, the cluster mean vector and the covariance matrix, for each of M clusters that are identified for the data. The classification algorithm then computes the probability of membership of h(i,j) in each of the M classes. The pixel is assigned to the class k which yields the highest probability P_k (h(i,j)). The equation used is

$$P_k\ (\ h(i,j)\ ) = b_k - 1/2\ (\ (h(i,j) - M_k)^T\ (\ C_k^{-1}\ )\ (h(i,j) - M_k)),$$

for class k, $0 \leq k \leq M-1$. M_k is the mean vector for class k, C_k is the covariance matrix for class k, and b_k is a predetermined value dependent on C_k. The superscript T denotes vector transpose.

Recall that the 512 X 512 image is distributed among the 1024 PEs such that each PE stores a 16 X 16 pixel portion of the image. Let the machine be partitioned into partitions of size M, and for simplicity, let $M = 2^r$. Within each partition, the PE addresses are designated PE 0, 1, ... , and M-1. PE(k) computes P_k for all pixels of the partition and has stored in its memory all constants needed for that computation, b_k, C_k, and M_k, $0 \leq k < M$ [GR76].

The algorithm computes P_k for all pixels of the 16 X 16 portion of the image in each PE. The first step calls on PE(i) to read $h^i(0,0)$ from memory, $0 \leq i < M$. PE(0) computes P_0 for the $h^0(0,0)$, PE(1) computes P_1 for the $h^1(0,0)$, and so forth. Assume that the intial class membership and its probability are zero. The new probability of class membership is compared to the current assigned class probability for $h^i(0,0)$. If the new probability is higher than the old, $h^i(0,0)$ is assigned to the new class. Next, $0 \leq k < M$, PE(k) sends its data to PE(k+1 modulo M), and the process is repeated. $h^i(0,0)$, its highest associated probabilitity, and the current class assignment are passed to all M PEs until all M class probabilities have been computed and the most probable class assigned, $0 \leq i < M$. Thus, the classes for M pixels have been computed. The process is repeated for the remaining pixels. Algorithm VII.4 describes the classification algorithm.

The algorithm is computationally intensive, since P_k is calculated for all $512^2 = 2^{18}$ pixels, for M values of k. A serial computer algorithm requires $O(M * d^2 512^2)$ time periods to compute the probability P_k for all k and for all pixels of the image. (Recall C_k^{-1} is a d x d matrix.) The algorithm above introduces parallelism to the task in several forms. First, it distributes the image across 1024 PEs. Next, the 1024 PEs are divided into 2^{n-r} partitions of size $M = 2^r$.

```
activate all PEs
for i = 0 step +1 until 15
    for j = 0 step +1 until 15
        class probability := 0;
        class := 0;
        current := $h^P(i,j)$; for PE(P), $0 \leq P < M$,
                                   within each partition.
** PE(k) calculates $P_k$ for the current pixel that is has. **
        for l = 0 step +1 until M-1
            $P_k$ := $b_k$ - 1/2 ((current - $M_k$)
                               * ($C_k^{-1}$) * (current - $M_k$));
            if ( $P_k$ > class probability) then
                                      class := k;
                                      class probability := $P_k$;
** Move the data to the next PE for the next $P_k$. **
            set ICN to PE +1 modulo M
            DTRin := (current, class, class probability);
            (current, class, class probability) := DTRout.
```

Algorithm VII.4: A parallel data classification algorithm.

Within each partition, M probabilities are computed at each step of the algorithm. The parallel algorithm requires $O(M * d^2 16^2)$ time periods to compute the probabilities. The speedup of the parallel algorithm over the serial algorithm is approximately three orders of magnitude.

All networks support this algorithm to some degree. For simplicity, assume that all d spectral observations of h(i,j) are transferred as one datum. From Chapter V, all networks except the Illiac network can partition the system under single control partitioning, as is required for the algorithm. The interconnection function needed is $f(x) = (x + 1)$ modulo M. Assume that the three data items to be passed, the current h(i,j), the class of h(i,j), and the associated class probability, are passed as a block of data whenever possible.

The recirculating Cube accomplishes f(x) using the following algorithm. Assume that all PEs in a partition have the same n-r most significant address bits.

```
for j = 0 step +1 until r-1
    MASK [$X^{n-r}X^{r-j}0^{j}$]
        $\text{Cube}_j$;
```

So, the recirculating Cube network requires r passes to implement f(x). A multistage Cube network implements f(x) in one pass.

The PM2I network can be partitioned such that all PEs in a partition have the same n-r least significant address bits. For $M = 2^r$, f(x) is realized by the interconnection function $PM2_{+(n-r)}$. The recirculating and multistage PM2I networks implement f(x) in one pass through the network. The pipelined multistage Cube and PM2I networks transfer the three data items as a block.

For $M > 2$, the Illiac network does not partition into complete Illiac networks of size M, as was seen in section V.3. If all PEs in a partition have consecutive PE addresses, then the Illiac_{+1} interconnection function routes the data for all PEs except PE(M-1) in one pass through the network. If $0 \leq M < \sqrt{N}$, then M-1 uses of Illiac_{-1} route the data from PE(M-1) to PE(0). The data transfer time is, thus,

$$M * 256 * (dr + dm + drn + dr)$$

$$+ M * 256 * (M-1) * (2 dr + dm + drn) =$$

$$M * 256 * (2 dr + dm + drn) (1 + M - 1)$$

$$= M^2 * 256 * (2 dr + dm + drn).$$

If $M \geq \sqrt{N}$, then data can be transferred from PE(M-1) to PE(0) in $1 + M/(\sqrt{N})$ steps. This requires using one Illiac_{+1} and $M/(\sqrt{N})$ $\text{Illiac}_{-\sqrt{N}}$ interconnection functions, as follows. Recall $M = 2^r$ and $M - 1 = 2^r - 1$. Then,

$$(2^r - 1) + 1 - (\sqrt{N}) * 2^r/(\sqrt{N}) = 0.$$

This yields a data transfer time of

$$M * 16^2 (Trc + (1 + M/(\sqrt{N})) * Trc)$$

$$= M * 16^2 [(M/(\sqrt{N}) + 2) Trc]$$

$$= M * 256 [(M/(\sqrt{N}) + 2) (2*dr + dm + drn)].$$

Clearly, the Illiac network is not as well suited to this algorithm as the other networks discussed.

Table VII.8 lists the data transfer times for these interconnection networks for the image classification algorithm. The discussion has shown that the data transfer time for this algorithm is not negligible. The best choice of interconnection network is the recirculating PM2I.

Table VII.8. Delay times for data transfer for the image classification algorithm, where M is the number of classes and D = dms = dm = dr = drn.

NETWORK	DELAY
recirculating Cube	$3\ M\ 256 \log_2 M\ (2\ dr + dm + drn) = 3072\ D\ M \log_2 M$
multistage Cube	$3\ M\ 256\ (2\ dr + 10\ dms) = 9206\ D\ M$
5-stage pipelined multistage cube	$M\ 256\ (dr + (5 + 3 - 1)\ (2\ dms + dr)) = 5632\ D\ M$
recirculating PM2I	$3\ M\ 256\ (2\ dr + dm + drn) = 3072\ D\ M$
combinational logic multistage PM2I	$3\ M\ 256\ (2\ dr + 10\ dms) = 9216\ D\ M$
5-stage pipelined multistage PM2I	$M\ 256\ (dr + (5 + 3 - 1)\ (2\ dms + dr)) = 5632\ D\ M$
Illiac	$M^2\ 256\ (2\ dr + dm + drn) = 1024\ D\ M^2,\ 0 \leq M < $ -/N. $M\ 256\ (M/($-/N$) + 2)\ (2\ dr + dm + drn) = 1024\ D\ M\ (M/32 + 2),\ M \geq $ -/N.

The worst choice is the Illiac network with its $O(M^2)$ contribution to the computation time of the algorithm. For Algorithm VII.1, as the number of pixels stored in a PE increased, the block size of the data transfer increase, and so, the pipelined networks transferred data faster than the unpipelined networks. For Algorithm VII.4, since data are not transferred in blocks, the phenomenon will not occur.

A classification algorithm can be designed which does not pass data among PEs. For such an algorithm, each PE calculates all M probabilities for all 16^2 which it stores in its memory. This method uses more memory than does Algorithm VII.4. Suppose that the data pixels are composed of four spectral observations. Then, the mean vector M has length four, and the covariance matrix is a 4 X 4 matrix. If the covariance matrix is symmetric, as is often the case, then only 10 of the 16 entries need be stored. So, for each class, a PE stores 10 words (for the covariance matrix) + 4 words (for the mean vector) + 1 word (for b_k) = 15 words. If M = 32, then each PE stores 480 more words for the algorithm which does not pass data than it does for Algorithm VII.4.

VII.6. Conclusions

In designing a parallel computer system, the selection of the interconnection network is dependent upon the types of algorithms the system will execute. This chapter has formulated three parallel picture processing tasks, a smoothing algorithm, a histogram formation algorithm, and a classification algorithm. For each, the impact of different interconnection networks on the efficiency of the algorithm has been explored.

For the parallel smoothing algorithm, data movement was in a neighborhood of eight PEs about any PE(i). Thus, the Illiac network and the recirculating PM2I network exhibited the fastest data transfer times. The recirculating Cube network had the slowest time, since a bit-serial addition or substraction algorithm was computed in order to control each data transfer.

Due to the recursive doubling algorithm, the data transfers for the parallel histogram formation algorithm are more global in nature. The data transfers are not always among PEs that are neighbors. So, the Illiac network requires the most time to transfer the data. The other recirculating networks can realize the interconnections in one pass through the network. Since the algorithm transfers data in blocks, the pipelined multistage networks pass the data in the least amount of time, and the combinational logic multistage networks are seen to be a poor choice for this algorithm.

The parallel data classification algorithm illustrates the use of single control partitioning. The 2^n PEs are partitioned into groups of M PEs each, where M is the number of data classes. Because the Illiac network does not partition into smaller, independent, complete networks for N > 2, this network performs poorly for the classification algorithm. The interconnection function needed for the algorithm is f(x) = (x+1) modulo M. For M a power of two, this function is one of the PM2I interconnection functions. Because of this, the recirculating PM2I network needs the least time to transfer data for this algorithm.

The analyses of Chapter IV were applied to three parallel image processing algorithms. The approach developed in this chapter can be generalized and applied to other algorithms. A given algorithm was analyzed for the data transfer needs, and the effects of specified interconnection networks on the performance of the algorithm were shown.

VIII. AN EMULATOR NETWORK FOR SIMD MACHINE INTERCONNECTION NETWORKS

VIII.1. Introduction

This chapter presents a single stage interconnection network, the emulator network, for SIMD computers. The emulator network can be used to simulate many types of interconnection networks in order to make informed choices for machine designs. In section 2, a hardware design of the emulator network is presented. A mechanism for defining control signals and transmitting data is discussed. Section 3 uses the design of section 2 to simulate some multistage interconnection networks discussed in the literature. These include Pease's indirect binary n-cube [PEA77], the STARAN network [BA76], the omega network [LAW75], and the data manipulator [FE74]. Section 4 considers the simulation of four recirculating networks, the Cube [SIE77a], the Shuffle-Exchange [ST71], the PM2I [SIE77a], and the Illiac [BOU72]. Section 5 shows how some useful permuations can be formed, and section 6 considers Benes-type networks. Section 7 discusses how the emulator network can operate in a machine that is partitioned into smaller SIMD machines, each of size a power of two. Section 8 shows how the emulator network can be used to simulate the interconnections in two multiple intruction stream -

multiple data stream computers, the CHoPP [SUB77] and the X-tree [DP78].

VIII.2. The Emulator Network

The emulator network, is a recirculating interconnection network that consists of the 2n-1 distinct PM2I functions and the Shuffle function. Figure VIII.1 shows a design for the emulator network as seen from PE(i). The emulator network can be built from N(2n+1) tristate buffers for each bit of network width. Each interconnection function corresponds to one of the 2n transmission buffers, and one buffer receives signals at the output. If the maximum delay of a buffer (such as the SN74240) is 7 ns, then the delay of the data transfer would be 14 ns plus the delay of the transmission lines between the transmitting and receiving buffers.

The emulator system can be visualized in two ways. Physically, it is a set of N PEs and the emulator network. Logically, it can be viewed as a set of N nodes, where node i consists of PE(i), input position i of the network (i.e., the 2n transmission buffers), and output position i of the network (i.e., the receiving buffer).

Each PE supplies the control bits for its part of the emulator network (i.e., PE(i) for node i) by means of a 2n bit Routing Control Register (RCR). For each pass through the network, each PE calculates where to send its data, then places the appropriate control bits in RCR. These bits go to the controls of the buffers at the input of the emulator network, each bit of the RCR corresponding to a different interconnection function. Figure VIII.2 shows the configuration of the emulator system.

A routing instruction is provided so that each PE can load its RCR with the appropriate controls for the given data transfer. For

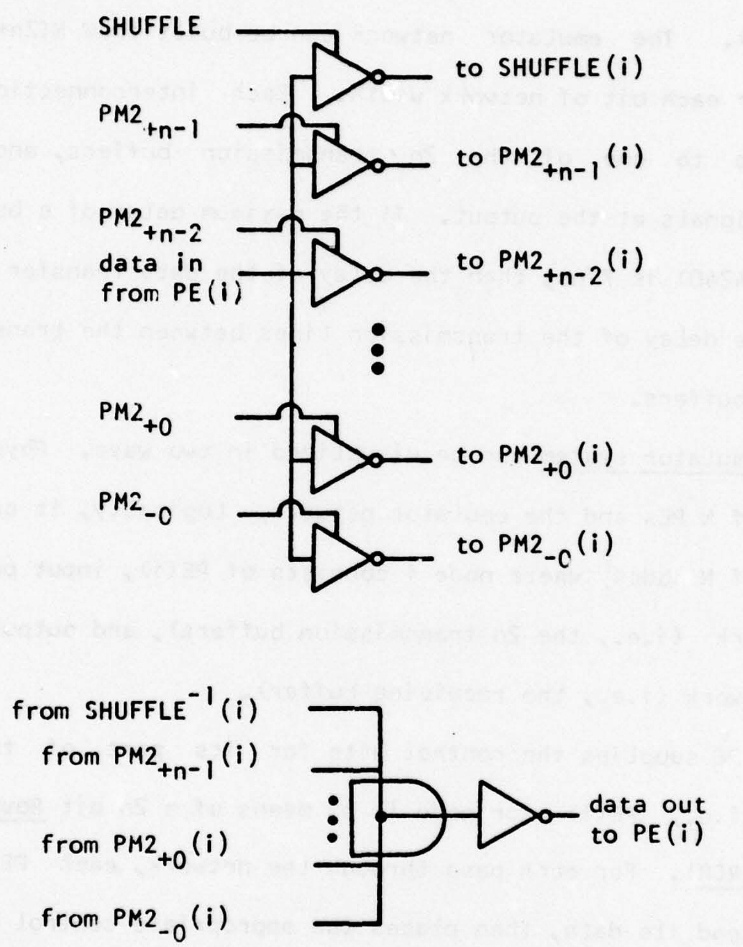


Figure VIII.1: A node of the emulator network for PE(i) for $N = 2^n$ PEs can be built using $N(2 \log_2 N + 1)$ tristate buffers. Control signals are provided by a register in PE(i).

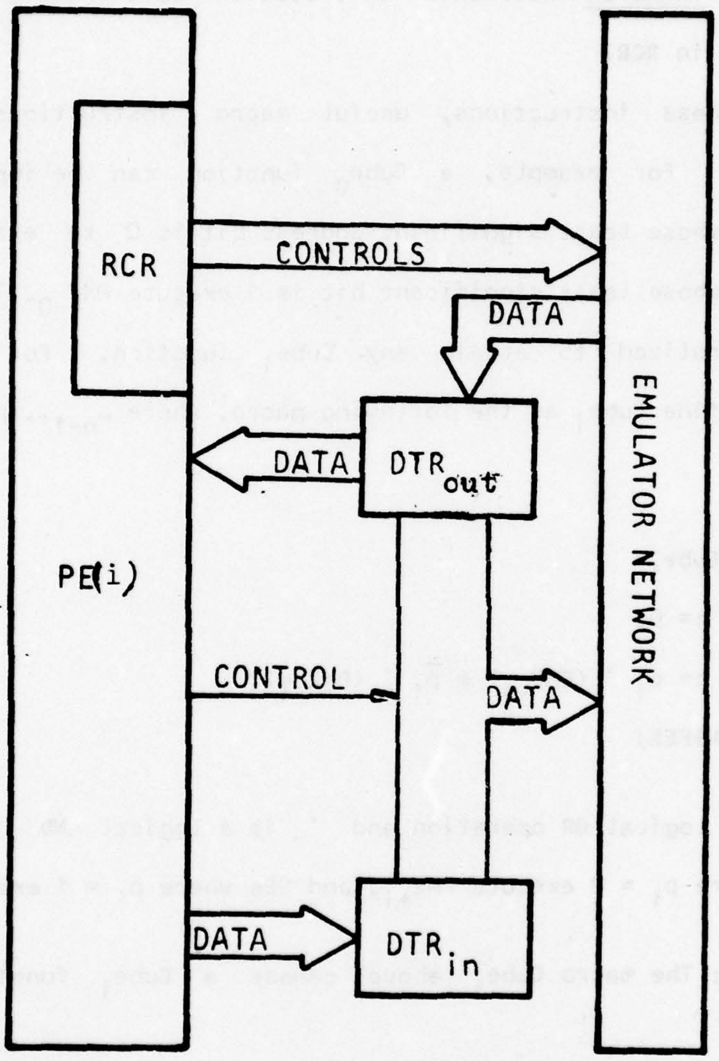


Figure VIII.2: Architecture of the Emulator System for PE(i).

simplicity, assume the instruction has 2n one-bit fields, one for each PM2I function and one for the Shuffle function. If a field is 0, the function is disabled. If a field is 1, the corresponding function is enabled. A <u>TRANSFER</u> instruction initiates the data transfer which uses the controls in RCR.

From these instructions, useful macro instructions may be constructed. For example, a Cube_0 function can be implemented by causing PEs whose least significant address bit is 0 to execute PM2_{+0} while PEs whose least significant bit is 1 execute PM2_{-0}. This process can be generalized to attain any Cube_i function. For notational purposes, define Cube_i as the following macro, where $p_{n-1} \ldots p_1 p_0$ is a PE address.

```
Define Cube_i;
    RCR := 0;
    RCR := $p_i \cdot (\text{PM2}_{-i}) + \overline{p_i} \cdot (\text{PM2}_{+i})$;
    TRANSFER;
```

where + is a logical OR operation and $\cdot$ is a logical AND. This means that PEs where $p_i = 0$ execute PM2_{+i}, and PEs where $p_i = 1$ execute PM2_{-i}.

<u>Lemma VIII.1</u>: The macro Cube_i above causes a Cube_i function to be executed.

<u>Proof</u>:

Case 1: PE(P), where $p_i = 0$. Then, PM2_{+i} routes the data to the PE whose address is $P + 2^i$ modulo $N = \text{Cube}_i(P)$.

Case 2: PE(P), where $p_i = 1$. Then, PM2_{-i} routes the data to the PE whose address is $P - 2^i$ modulo $N = \text{Cube}_i(P)$.

For simplicity in the algorithms below, the following assumptions are made:

(1) data to be transferred from PE(i) to PE(j) start in the DTRin of PE(i) and must end in the DTRin of PE(j), $0 \leq i, j < N$; and

(2) the network always takes the data in DTRin as its input.

VIII.3. Simulation Results: Multistage Networks

The indirect binary n-cube network [PEA77] (Figure III.5) is based on the Cube interconnection functions. In simulating this network, the PEs of the emulator network can calculate controls at each stage and apply one of the Cube_i macros defined above.

Theorem VIII.1: The emulator network can simulate the n-stage indirect binary n-cube network in n passes.

Proof: By definition, stage i, $0 \leq i < n$, of the indirect binary n-cube network executes Cube_i with individual box control and two function interchange boxes. In [PEA77], it is proposed that a microprogrammable microcomputer called a switch controller generate the settings for the interchange boxes. Each switch controller provides controls for a "set" of interchange boxes. Each control is either "exchange" or "straight." The program which the switch controller uses to generate the controls could be run in each PE of the emulator system in order to simulate the effects of the switch controller. Since each PE need compute only the control, c, for node i, the program run is a subtask of the program that the switch controller would use.

An algorithm that uses the emulator network to simulate an indirect binary n-cube is

```
activate all PEs,
RCR := 0;
for i = 0 step +1 until n-1 do
    calculate interchange box control c;
    if c = "Exchange" then $\text{Cube}_i$;
```

DTRin := DTRout.

Cube_i is the macro instruction defined in section VIII.2. Since stage i of the indirect binary n-cube executes Cube_i, $0 \leq i < n$, the results of the algorithm are the same as the actions of the indirect binary n-cube. The action of the emulator network nodes j and $j+2^i$ simulate the action of the interchange box at stage i whose inputs are j adn $j+2^i$, $0 \leq j < N$, and the i^{th} bit of j is 0. Because data in the n-cube travel through stage 0, then stage 1, etc., the loop of the algorithm goes from i = 0, to i = 1, etc. []

Banyon networks are a class of interconnnection networks which are defined in terms of graph representations. The graph of a banyon network is a Hasse diagram of a partial ordering in which there is one and only one path from any input vertex to any output vertex [GL73]. F, the fanout, is the number of arcs incident into each vertex at a stage from the last stage, and S, the spread, is the number of arcs incident out from each vertex to the next stage. For S = F = 2, each vertex of the graph represents a 2 X 2 crossbar switch, i.e. an interchange box. One type of banyon structure is the SW banyon, shown in Figure VIII.3 with its corresponding graph for N = 8.

Corollary VIII.1: The emulator network can simulate an n-stage SW banyon network for S = F = 2 in n passes.

Proof: From [GL73], the recursive definition of a banyon is as follows:

(1) A one level SW banyon structure with fanout F and spread S is a crossbar with F bases and S apexes.

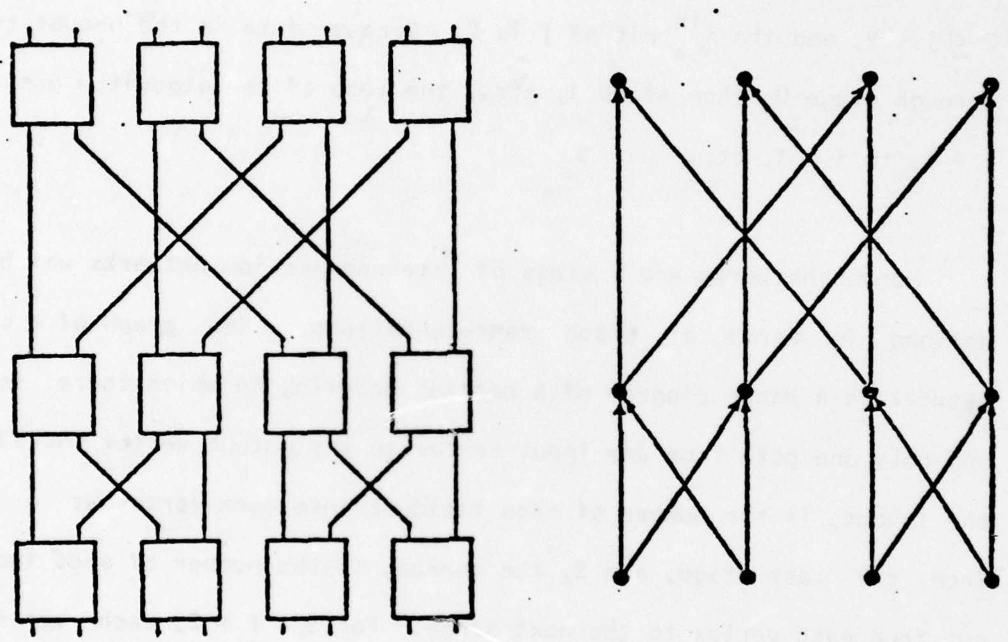


Figure VIII.3: An SW banyon network and its graph for N = 8 [GL73].

(2) An L-level SW banyon structure with fanout F and spread S can be synthesized by interconnecting S^{L-1} crossbars and F identical (L-1) level SW banyon structures, all with fanout F and spread S. The apexes of the SW structures are connected to the bases of the crossbars such that the interconnection graph is a crossbar. Also, each crossbar must be connected to every component SW structure in the same way, i. e., if it is connected to the i^{th} apex of one SW, it must be connected to the i^{th} apex of each of the others.

So, an SW banyon with S = F = 2 has the same topology as the n-stage indirect binary n-cube network. From Theorem VIII.1, the emulator network can simulate an n-stage SW banyon network for S = F = 2 in n passes through the network. []

The STARAN flip network (Figure III.4) is also a multistage cube-type network. The emulator system simulates the STARAN flip network with either flip or shift controls with the aid of the Cube_i macros and the computation facilities of the PEs.

<u>Corollary VIII.2</u>: The emulator network can simulate the n-stage STARAN flip network with flip or shift controls in n passes.

<u>Proof</u>: The STARAN flip network [BA76] has the same topology as the indirect binary n-cube with two function interchange boxes, but has a different control structure [SIS78]. The flip control for the STARAN flip network is individual stage control. For the flip control, the STARAN uses the control vector $F = (f_{n-1} \dots f_0)$, where $f_i = 1$ causes stage i to execute Cube_i. An algorithm using the emulator network to simulate the STARAN network under flip control is

```
activate all PEs,
RCR := 0;
CU (control unit) broadcasts F to all PEs;
for i = 0 step +1 until n-1 do
    if $f_i$, then $Cube_i$;
             DTRin := DTRout.
```

The shift controls for STARAN provide i+1 control signals at stage i in order to perform shifts of 2^m modulo 2^p, $0 \leq m \leq p \leq n$. Figure III.4b shows the shift controls for the STARAN network for N = 8. Each control signal is one bit, where 0 indicates straight and 1 indicates $Cube_i$ at stage i. Recall the definition of the shift control signals from Chapter V. At stage i, let the i+1 control signals be labeled ci0, ci1, ..., cii. Stage 0 has one control, c00, that affects all PEs. The signals control groups of PEs in the following manner at stage i, i > 0.

ci0: $X^{n-i}0^i$;

ci1: $X^{n-i}0^{i-1}1$;

ci2: $X^{n-i}0^{i-2}1X$;

. . .

cii: $X^{n-i}1X^{i-1}$.

Assume that the shift controls can be broadcast to the PEs as a vector, c_i, of length n that is padded with zeroes, as needed. For example, for N = 64, c_3 = (0 0 0 c33 c32 c31 c30). Let each PE precompute and store an n bit mask for stage i, $MASK_i$, which has "1" in position j if the PE is affected by cij, $0 \leq i < n$. For example, let N = 64, and n = 6. For stage 0, all PEs are affected by c00. So, $MASK_0$ = (0 0 0 0 0 1). For stage 1, all even PEs are affected by c10.

For these PEs, $MASK_1$ = (0 0 0 0 0 1). All odd PEs are affected by c11, and so for these PEs, $MASK_1$ = (0 0 0 0 1 0). If the data words of the system are 16 bits long, then for $N \leq 2^{16} = 65,536$, only n words of storage are needed. To compute the control at stage i for a PE, $MASK_i$ is ANDed with c_i. If the result is not zero (all 0's), then $Cube_i$ is executed by the PE. An algorithm that uses the emulator network to simulate the STARAN network with shift controls is

```
    activate all PEs,
    RCR := 0;
    for i = 0 step +1 until n-1 do
        CU broadcasts stage i controls, $c_i$, to all PEs;
        CNTRL := $c_i \cdot MASK_i$;
        if NOT( CNTRL = 0 ) then $Cube_i$;
                                    DTRin := DTRout.
```

The shifts of 2^m modulo 2^p could be more efficiently implemented using the PM2I functions of the emulator network. For example, the shifts 2^m modulo 2^n are the PM2I functions $PM2_{+m}$. However, the algorithm above more closely simulates the actions of the STARAN flip network. As in the proof of Theorem VIII.1, the action of emulator nodes j and $j+2^i$ simulate the action of the interchange box at stage i whose inputs are j and $j+2^i$, $0 \leq j < N$, and the i^{th} bit of j is 0. Because data in the flip network pass first through stage 0, then stage 1, etc., the loop index i in the algorithm goes from 0 to n-1. []

The generalized cube network [SIS78] (Figure III.6) is defined as a standard topology for multistage Cube networks, and may be used with either type of interchange box and any of the three multistage control structures. The PM2I interconnection functions of the emulator network can be used to simulate the broadcast states of four function interchange boxes. For example, an upper broadcast from PE(0) to PE(1) is accomplished by saving the data in PE(0) and also routing it to PE($PM2_{+0}$ (0)) = PE(1).

Theorem VIII.2: The emulator network can simulate the n-stage generalized cube network in n passes.

Proof: Assume the most flexible construction and control, that of four function interchange boxes and individual box control. In [SIS78], no particular method of generating control signals was discussed. Assume that each PE in the emulator system can generate a pair control bits, call them c, for each pass through the emulator network (one stage of the generalized cube). An algorithm which uses the emulator network to simulate a generalized cube is as follows, where p_i is the i^{th} bit of the PE address.

```
activate all PEs,
RCR := 0;
for i =  n-1 step -1 until 0 do
   if $p_i$ = 0 and c = ("exchange" or "upper broadcast")
                        then RCR := $PM2_{+i}$;
   if $p_i$ = 1 and c = ("exchange"  or  "lower broadcast")
                        then RCR := $PM2_{-i}$;
```

```
TRANSFER;

DTRin := DTRout.
```

After n steps, the algorithm routes the data in the same way as the generalized cube. As in the proof of Theorem VIII.1, emulator nodes j and $j+2^i$ act as an interchange box. Since the data pass first through stage n-1, then stage n-2, etc., the loop index i goes from n-1 to 0. []

Each stage of an n-stage Shuffle-Exchange network is a shuffle interconnection possibly followed by an exchange (Figure III.3), depending on the control settings.

Corollary VIII.3: The emulator network can simulate the n-stage Shuffle-Exchange network in n passes.

Proof: The n-stage Shuffle-Exchange network is topologically equivalent to the generalized cube [SIS78]. Using Theorem VIII.2, this network can be simulated in n steps. []

Lang and Stone [LAST76] have presented a simplified control scheme for a Shuffle-Exchange network with two function interchange boxes. The control bits for the interchange boxes are calculated from those of the previous stage. For N = 8, Figure VIII.4 shows an expanded Shuffle-Exchange network with this simplified control. Let the first stage of the network be labeled stage n-1. For i even, $C^k(i)$ is the control for interchange box for data lines i and i+1 at stage n-k, where $C^k(i) = 1$ means exchange, and $C^k(i) = 0$ means straight, $1 \leq k \leq n$. Note that $C^k(i) = C^k(i+1)$. For i even and $k \geq 2$, the control at stage n-k is

$$C^k(i) = C^{k-1}(\text{Shuffle}^{-1}(i)) \circ C^{k-1}(\text{Shuffle}^{-1}(i+1))$$

where o is a particular boolean operation. The initial condition, $C^1(i)$, is

$$C^1(i) = D(\text{Shuffle}^{-1}(i))_{n-1}$$

for i even, where $D(j)_{n-1}$ is the most significant bit of the destination of data from PE(j). As an example, let N = 8 and let the boolean operation be the exclusive or function. Figure VIII.5 realizes a cyclic shift of 3, that is, PE(x) sends data to PE(x + 3 modulo 8). The initial conditions are

$C^1(0) = 0$,

$C^1(1) = 0$,

$C^1(2) = 1$,

$C^1(3) = 1$,

$C^1(4) = 1$,

$C^1(5) = 1$,

$C^1(6) = 1$,

$C^1(7) = 1$.

Lang has shown that if the boolean function for the simplified control is the exclusive or function, then all uniform shifts of PE(x) to PE(x + j modulo N) can be accomplished by the network. If the equivalence function (f(x,y) = 1 if and only if x = y) is used, then data can be passed from PE(x) to PE((t - x) modulo N), $0 \leq x < N$, t an integer.

In [LAST76], it is proposed that the boolean function is hardwired into the control structure of the network. The emulator control is not bound by the hardware in this way, and the PEs can choose any boolean

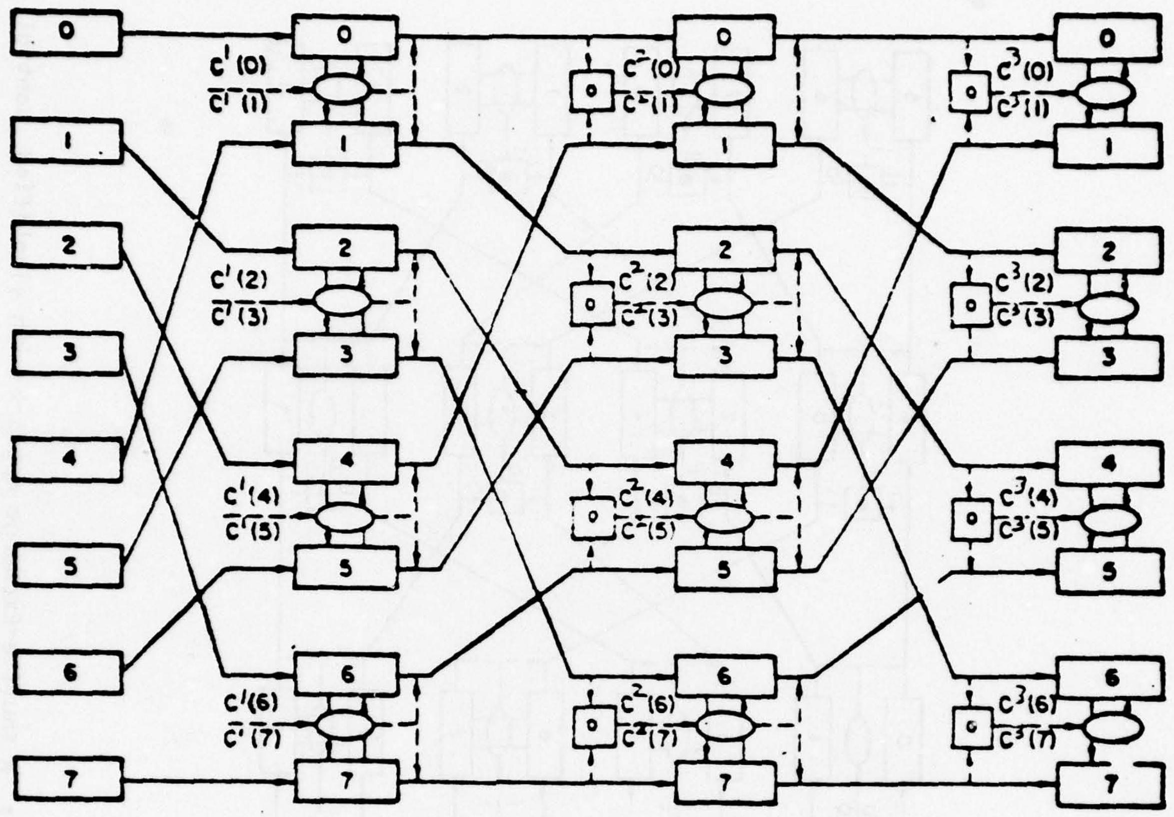


Figure VIII.4: A Shuffle-Exchange network with simplified control for N = 8. Dotted Lines represent paths of control signals. The control bits for stage i are calculated from those of the previous stage. [LAST76]

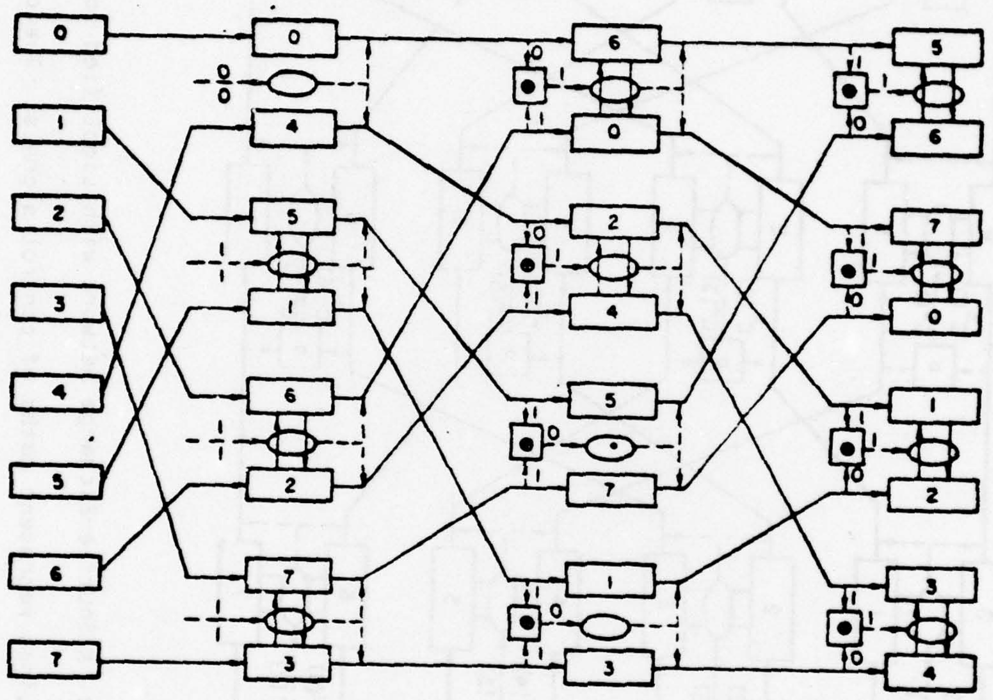


Figure VIII.5: A Shuffle-Exchange network with simplified control can realized a cyclic shift of three when the specified boolean function for calculating the control bits is the exclusive or function. [LAST76]

function to be implemented. Thus, the effects of different operators may be simulated.

<u>Corollary VIII.4</u>: The emulator network can simulate the n-stage Shuffle-Exchange network with simplified controls in n passes.

<u>Proof</u>: Assume that the control bits at each stage are passed with the data they affected, as in [LAST76]. At any stage, for each datum, call this bit C. After a Shuffle transfer at stage k, C is either $C^{k-1}(\text{Shuffle}^{-1}(i))$ or $C^{k-1}(\text{Shuffle}^{-1}(i+1))$, as mentioned above (Figure VIII.4), for i even, $k \geq 2$. Assume that a destination $D_j = d_{n-1}\ldots d_0$ is initially specified for data from PE(j). Since the multistage Shuffle-Exchange network is topologically equivalent to the generalized cube network, the algorithm below is based on Theorem VIII.2. This could be done in n passes, as follows. Let each PE keep a copy of the data as well as send it to the PE that differs in the i^{th} bit position. If the control bit is 0, the retained data is moved to DTRin. For clarity, the following algorithm simulates the Shuffle-Exchange network with simplified control in 2n passes.

```
activate all PEs,
RCR := 0;
DATA := DTRin;
*** Initialize controls ***
    C := $d_{n-1}$;
    if $p_{n-1}$ = 0 then DTRin := C;
                            RCR := $PM2_{+(n-1)}$;
                            TRANSFER;
```

```
        if $p_{n-1}$ = 1 then C := DTRout;
        DTRin := DATA;
        if C then $Cube_{n-1}$;
        DATA := DTRout;
        for k = 2 step +1 until n do
    *** Pass controls for adjacent PEs. ***
            DTRin := C;
            $Cube_{n-k}$;
            C1 := DTRout;
    *** Calculate new control. ***
            $C^k$ := C o C1;
            C := $C^k$;
            DTRin := DATA;
            if C then $Cube_{n-k}$;
            DATA := DTRout;
        DTRin := DATA.
```

If the data item and the C bit requires more bits than the network width, multiples of n passes will be needed. []

The omega network [LAW75] is an n-stage Shuffle-Exchange network. One method of controlling the omega network is by associating with each datum the address of the destination PE. This address is called a <u>destination tag</u>. The omega network considers only two function interchange boxes for this type of control. Let a destination tag $D_i = d_{n-1}...d_0$ be passed with the datum originally from PE(i), $0 \leq i < N$. At stage j, if $d_j = 0$ for the upper input of the interchange box, the interchange box is set to straight. If $d_j = 1$ for the upper input, the interchange box is set to exchange. The jth bit of the destination tags of the lower and upper inputs must be complements, or else an error occurs.

<u>Theorem</u> <u>VIII.3</u>: Let the data word to be transfered be W bits wide, and let a destination tag associated with the data word be n bits wide. Consider an n stage omega network with a width of W+n bits. The emulator network can simulate an n stage omega network with destination tags for control in n passes.

<u>Proof</u>: The omega network is topologically equivalent to the generalized cube [SIS78], and an algorithm based on Theorem VIII.2 simulates the omega network in n passes.

```
activate all PEs,
RCR := 0;
for i = n-1 step -1 until 0 do
    if $p_i = \overline{d}_i$ then Cube$_i$;
                        DTRin := DTRout.
```

If the width of the network is less than the length of D_i plus the length of the data, multiples of n passes through the network are needed. []

<u>Theorem</u> <u>VIII.4</u>: The emulator network can simulate the n-stage omega network in n passes.

<u>Proof</u>: The omega network is an n-stage Shuffle-Exchange network with four function interchange boxes [LAW75]. In [LAW75], no control scheme

for four function interchange boxes is discussed. Assume the control convention used in Theorem VIII.2. Since the omega network is topologically equivalent to a generalized cube network with four function interchange boxes and individual cell control [SIS78], the algorithm to simulate the omega network is the same as the one in Theorem VIII.2.

If j passes through the omega network are required to realize a given data transfer, then j*n passes through the emulator network realize the same transfer. □

The data manipulator [FE74] is an n-stage PM2I network where stage i performs $PM2_{+i}$, $PM2_{-i}$, or straight. For the control of the network discussed in [FE74], each stage of the network is controlled by a pair of signals, specifying two of H_1, H_2, U_1, U_2, D_1, and D_2 as defined in Chapter III and shown in Figure III.7.

Theorem VIII.5: The emulator network can simulate the n-stage data manipulator network in n passes.

Proof: The control signals for the data manipulator are supplied for each stage by external control. With the emulator network, at pass k, PE(j) controls cell j of stage n-k. For each pass, call the two control signals supplied to the stage C1 and C2, and let them be broadcast to all PEs. To simulate the data manipulator network with the emulator:

```
activate all PEs,
RCR := 0;
for i = n-1 step -1 until 0 do
    clear RCR;
    DTRout := DTRin;
    CU broadcasts C1 and C2;
    if $p_i$ = 0 then
                    if C1 = U then RCR := $PM2_{-i}$;
                    if C1 = D then RCR := RCR + $PM2_{+i}$;
               else if C2 = U then RCR := $PM2_{-i}$;
                    if C2 = D then RCR := RCR + $PM2_{+i}$;
    TRANSFER;
    DTRin := DTRout.                                        []
```

The augmented data manipulator (ADM) network [SIS78] is based on Feng's data manipulator [FE74]. It allows each of the N cells at each stage to receive its own control signals, any combination of H, U, and D.

<u>Corollary</u> <u>VIII.5</u>: The emulator network can simulate the n-stage augmented data manipulator (ADM) network in n passes.

<u>Proof</u>: No particular method of generating the control signals for the ADM is specified in [SIS78]. It is assumed that for each data item the encoded control information for stage i, C(i), is passed with the data, and can be any combination of one to three of H, U, and D.

```
activate all PEs,
RCR := 0;
for i = n-1 step -1 until 0 do
     clear RCR;
     DTRout := DTRin;
     if C(i) = U then RCR := $PM2_{-i}$;
     if C(i) = D then RCR := RCR + $PM2_{+i}$;
     TRANSFER;
     DTRin := DTRout.
```

If the data item and the n C(i)'s require more bits than the width of the network, multiples of n passes will be required. If j passes through the ADM are required to realize a given data transfer, then j*n passes through the emulator network can realize the same data transfer[]

A CC banyon network can be laid out as a crosshatch pattern on the surface of a cylinder. CC is an acronym for Cylindrical Crosshatch [GL73]. Figure VIII.6 shown a three stage CC banyon network for N = 8 and S = F = 2.

<u>Theorem</u> <u>VIII.6</u>: The emulator network can simulate an n-stage CC banyon network for S = F = 2 in n passes.

<u>Proof</u>: A CC banyon network [GL73], with fanout F and spread S, is defined as an L stage network with $N = S^L$ vertices at each level. There is an arc from vertex Vi_k at stage k to vertex Vj_{k+1} at stage k+1 if and only if $j = (i + m * S^k)$ modulo N for some m, $0 \leq m < S$. For S = 2, m $\in$ {0,1}. Thus, at stage k, since F = 2, each vertex Vi_k has connections

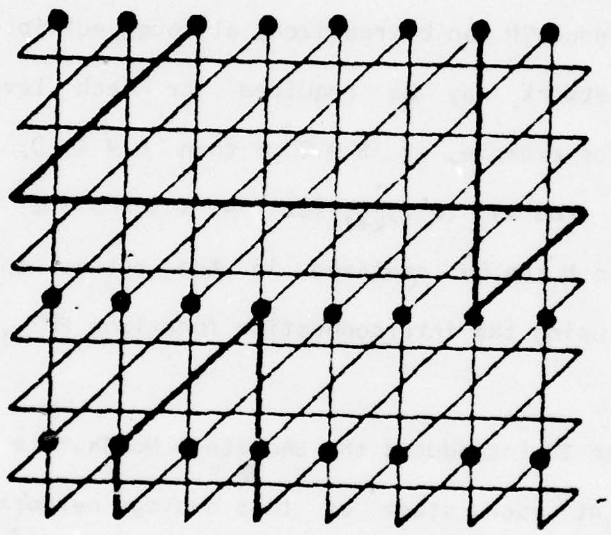


Figure VIII.6: The graph of a CC banyon network for S = F = 2 and N = 8 [GL73] connects Vi_k at stage k to Vj_k at stage k+1 if and only if j = i or j = $(i + 2^k)$ modulo N, $0 \leq k < n$.

to Vi_{k+1} and to Vj_{k+1} where $j = (i + 2^k)$ modulo N. Since the emulator network provides all $PM2_{+i}$ functions, $0 \leq i < n$, it can simulate the CC banyon network for $S = F = 2$. []

<u>Corollary</u> <u>VIII</u>.<u>6</u>: The emulator network can simulate any CC banyon for $S = F = 2^s$ and $N = S^L$.

<u>Proof</u>: For $S = 2^s$, $m \in \{ 0, 1, \ldots, 2^s - 1 \}$. The emulator provides all PM2I functions, and so the connections from Vi_k to Vj_{k+1} for $j = (i + m * S^k)$ modulo N can be realized, although multiple passes through the emulator network may be required for each level of the CC banyon network. For example, if $S = 2^2$, then $m \in \{ 0, 1, 2, 3 \}$. The connection from Vi_k to Vj_{k+1} for $j = (i + 3 * 2^2)$ modulo $N = (i + 8 + 4)$ modulo N can be realized in two passes through the emulator network by using the interconnection functions $PM2_{+3}$ and $PM2_{+2}$. []

Chapter IV introduced the Shuffle - No Shuffle - Exchange (SNSE) network. At each stage of this n-stage network, the options are no change, Shuffle, Exchange, or Shuffle-Exchange. This network can perform 1 to n-1 Shuffles in one pass through the network, as well as perform all the data transfers of a multistage Shuffle-Exchange network.

<u>Theorem</u> <u>VIII</u>.<u>7</u>: The emulator network can simulate an n-stage SNSE network in at most 2n passes.

<u>Proof</u>: An algorithm which simulates the SNSE network is

```
activate all PEs,
RCR := 0;
for i = n-1 step -1 until 0 do
    CU broadcasts SNSE controls;
    if (Shuffle or Shuffle-Exchange) then RCR := Shuffle;
                                          TRANSFER;
                                          DTRin := DTRout;
    if (Exchange or Shuffle-Exchange) then Cube$_0$;
                                           DTRin := DTRout.      []
```

The algorithm above requires at most 2n passes to transfer the data. If the SNSE controls specify no shuffle and no exchange, then the emulator network need not pass data. If the SNSE controls specify a shuffle and an exchange at each stage, then the emulator network used 2n passes to accomplish the data transfer.

VIII.4. Simulation Results: Recirculating Networks

Chapter II defined a recirculating network as an interconnection network that uses a single stage of switches to route data among PEs. Chapter IV presented hardware designs and data transfer time comparisons for several recirculating networks, in particular, the Shuffle-Exchange, the Cube, the PM2I, and the Illiac. The emulator network can simulate these recirculating networks.

Theorem VIII.8: The emulator network can simulate a recirculating Shuffle-Exchange network.

Proof: A recirculating Shuffle-Exchange network allows each PE to execute either a Shuffle interconnection function or an Exchange interconnection function for each transfer [SIE77a]. The Shuffle is one of the interconnection functions provided by the emulator network, and the Exchange = $Cube_0$. []

Theorem VIII.9: The emulator network can simulate a recirculating Cube network.

Proof: In a recirculating Cube network, all active PEs execute the $Cube_i$ interconnection function that is broadcast by the control unit [SIE77a]. The emulator network can execute the macro instruction $Cube_i$ (Lemma VIII.1). []

Theorem VIII.10: The emulator network can simulate the recirculating PM2I network.

Proof: In a recirculating PM2I network, all active PEs execute the single PM2I interconnection function that is broadcast by the CU [SIE77a]. The 2n-1 distinct PM2I functions are included in the emulator network.[]

Corollary VIII.7: The emulator network can simulate the Illiac network.

Proof: The four Illiac interconnection functions [BOU72] are a subset of the PM2I functions, where Illiac_{+1} (P) = PM2_{+0} (P), Illiac_{-1} (P) = PM2_{-0} (P), $\text{Illiac}_{+\sqrt{N}}$ (P) = $\text{PM2}_{+n/2}$ (P), and $\text{Illiac}_{-\sqrt{N}}$ (P) = $\text{PM2}_{-n/2}$ (P). []

VIII.5. Other Useful Permutations

The emulator network can simulate the perfect shuffle of size 2^{n-k} and the inverse perfect shuffle of size 2^{n-k}, for any $0 \leq k < n$. Figure VIII.7 illustrates the perfect shuffle and its inverse for $N = 8$.

Theorem VIII.11: The emulator network can simulate an inverse perfect shuffle of size 2^{n-k} in $n-k-1$ passes, $0 \leq k < n$.

Proof: For $PE(P) = p_{n-1} \cdots p_0$, at step i, the simulation algorithm has the effect of comparing bits p_i and p_{i+1} and, if different, sending the data to PE $p_{n-1} \cdots p_{i+2} p_i p_{i+1} p_{i-1} \cdots p_0$.

```
for i = 0 step +1 until n-k-2 do
    if $p_i \neq p_{i+1}$ then RCR := $p_i \cdot (PM2_{+i}) + \overline{p}_i \cdot (PM2_{-i})$;
    TRANSFER;
    DTRin := DTRout.
```

The case $p_i \cdot (PM2_{+i})$ sends data from PE $p_{n-1} \cdots 01 p_{i-1} \cdots p_0$ to $p_{n-1} \cdots 10 p_{i-1} \cdots p_0$. The case $\overline{p}_i \cdot (PM2_{-i})$ sends data from PE $p_{n-1} \cdots 10 p_{i-1} \cdots p_0$ to $p_{n-1} \cdots 01 p_{i-1} \cdots p_0$. Thus, after $n-k-1$ comparisons, the data originally in $P = p_{n-1} \cdots p_1 p_0$ is transferred to PE $p_{n-1} \cdots p_{n-k} p_0 p_{n-k-1} \cdots p_1$. An inverse perfect shuffle has been accomplished. []

Theorem VIII.12: The emulator network can simulate a perfect shuffle of size 2^{n-k} in $n-k-1$ passes, $0 \leq k < n$.

Proof: The simulation algorithm is similar to that of Theorem VIII.11. Adjacent bits of the PE address are compared and, if different, the data

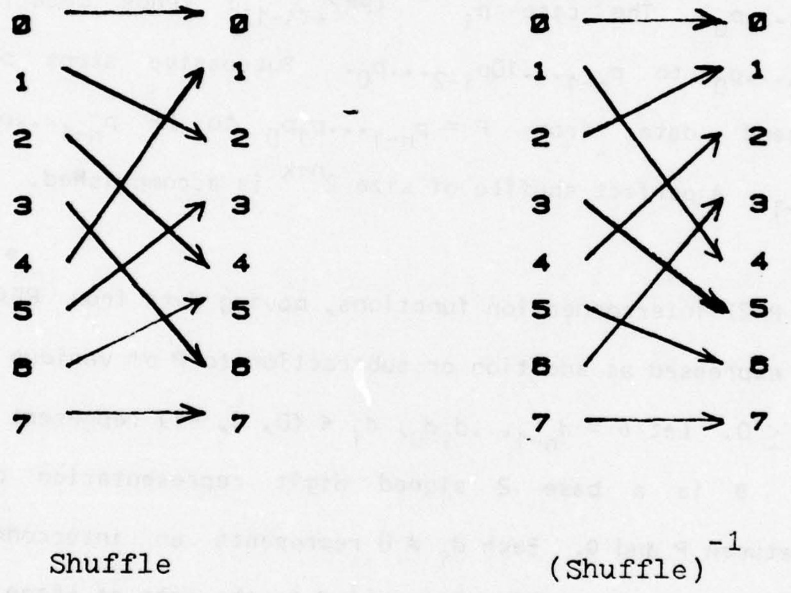


Figure VIII.7: The perfect shuffle and the inverse perfect shuffle for N = 8.

is routed as follows.

```
RCR := 0;
for i = n-k-1 step -1 until 1 do
    if $p_{i-1} \neq p_i$ then RCR := $p_i \cdot (PM2_{-(i-1)}) + \overline{p}_i \cdot (PM2_{+(i-1)})$;
    TRANSFER;
    DTRin := DTRout.
```

The case $p_i \cdot (PM2_{-(i-1)})$ sends data from PE $p_{n-1}\cdots10p_{i-2}\cdots p_0$ to $p_{n-1}\cdots01p_{i-2}\cdots p_0$. The case $\overline{p}_i \cdot (PM2_{+(i-1)})$ sends data from PE $p_{n-1}\cdots01p_{i-2}\cdots p_0$ to $p_{n-1}\cdots10p_{i-2}\cdots p_0$. Successive steps of the algorithm send data from $P = p_{n-1}\cdots p_1p_0$ to PE $p_{n-1}\cdots p_{n-k}p_{n-k-2}\cdots p_0p_{n-k-1}$. A perfect shuffle of size 2^{n-k} is accomplished. □

For the PM2I interconnection functions, moving data from PE(P) to PE(Q) can be expressed as addition or subtraction to P of various values of 2^i, $n < i \leq 0$. Let $D = d_{n-1}\cdots d_1d_0$, $d_i \in \{0, 1, -1\}$ represent these transitions. D is a base 2 signed digit representation of the difference between P and Q. Each $d_i \neq 0$ represents an interconnection function. If $d_i = 1$, then $PM2_{+i}$ is applied to the data at stage i. If $d_i = -1$, then $PM2_{-i}$ is applied to the data at stage i. If $d_i = 0$, then $loc_{n-i}(P) = loc_{n-i-1}(P)$.

Using the PM2I functions of the emulator network, there is more than one path from P to Q, and so D is not a unique number. For example, for N = 16, there are four different signed digit representations of the path from 0 to 7. They are

a) 0 1 1 1

b) 1 0 0 -1

c) 1 0 -1 1

d) 1 -1 1 1.

The number of non-zero digits in each of these representations is called the multiplicity. Form b has the lowest multiplicity (two) and is called the n-digit minimal form of natural value k, for $n = 4$ and $k = 7$.

Reitwiesner [REI60] has presented a detailed study of the existence and generation of minimal forms. These results are summarized here. Let $0x_{n-1}\ldots x_1x_0$ be an n+1 digit binary number of natural value k. Then,

1) there exists a unique canonical form $y_ny_{n-1}\ldots y_0$ in signed digit notation such that

$$y_i * y_{i-1} = 0,\ 0 < i < n,$$

and,

2) no other form of this natural value k has lower multiplicity than the canonical form.

A serial canonical recoding algorithm gives the following rules for converting $X = x_{n-1}\ldots x_0$ to $Y = y_n\ldots y_0$, $y_i \in \{0, 1, -1\}$. Beginning with $i = 0$, inspect x_i, x_{i+1}, and a carry digit c_i. Use the results to generate y_i and a carry out c_{i+1} to be used at the next step. The rules are

```
$c_0$ := 0;
$x_n$ := 0;
$x_{n+1}$ := 0;
for i = 0 step +1 until n
        $c_{i+1}$ := 1 if $((x_{i+1} + x_i + c_i) > 1)$
                   0 otherwise
        $y_i := x_i + c_i - 2c_{i+1}$;
```

The symbol "+" refers to addition. For example, the algorithm has the effect of replacing the string 011...110 in X by the string 100...0-10 in Y.

In order to move data from PE(P) to PE(Q), consider the distance D between them such that (P + D) modulo N = Q. Let D be a positive binary number $d_{n-1}...d_0$ that represents a path from P to Q utilizing only the $PM2_{+i}$ interconnection functions.

<u>Lemma VIII.2</u>: Given D, a path utilizing the minimum number of PM2I interconnection functions from P to Q can be found by recoding D into a signed digit vector in canonical form.

<u>Proof</u>: Apply the canonical recoding algorithm above to D to yield $y_n y_{n-1} ... y_0$. By [REI60], this result is unique and has the least multiplicity of any form of the natural value represented by D. The path from P to Q is represented by

$$(P + y_n 2^n + y_{n-1} 2^{n-1} + ... + y_0 2^0) \text{ modulo } N =$$
$$(P + y_{n-1} 2^{n-1} + ... + y_0 2^0) \text{ modulo } N = Q. \qquad []$$

<u>Theorem VIII.13</u>: The emulator network performs a uniform shift of

distance A, i.e., PE(P) sends data to PE(P+A modulo N) for all P, $0 \leq P < N$, in K passes through the network, where K is the number of nonzero digits in the canonical signed digit form of A.

<u>Proof</u>: The distance that all PEs transfer data is A, and from Lemma VIII.2 A the can be recoded into canonical signed digit form that has the fewest number of nonzero digits of any binary representation of A. Let the recoded A be $y_{n-1}y_{n-2}\ldots y_1y_0$. The following algorithm performs a uniform shift of A.

```
activate all PEs,
for i = n-1 step -1 until 0
    DTRout := DTRin;
    clear RCR;
    if $y_i$ = 1 then RCR := $PM2_{+i}$;
    if $y_i$ = -1 then RCR := $PM2_{-i}$;
    TRANSFER;
    DTRin := DTRout;
```
[]

<u>Theorem VIII.14</u>: The emulator network can perform a broadcast function in $|n/2|$ passes through the network.

<u>Proof</u>: The maximum number of interconnection functions that must be applied to send data from PE(P) to any arbitrary PE(Q) can be found by examining the distance between P and Q. By the canonical recoding algorithm, any distance D between P and Q can be recoded into signed digit form with minimal multiplicity, and thus the minimal number of interconnection functions. This result, Y, has the property,

$$y_i * y_{i-1} = 0,\ 0 < i < n.$$

For arbitrary Q, the maximum multiplicity of Y must occur for $Y = (01)^{n/2}$ or $(10)^{n/2}$ for n even, and for $Y = 1(01)^{(n-1)/2}$ for n odd. The maximum multiplicity if any arbitrary D is $|n/2|$. Thus, an upper bound for the maximum number of interconnection functions that must be applied to move data from P to Q is $|n/2|$.

From this, then, the emulator network requires $|n/2|$ passes through the network to execute a broadcast function. At the first pass, source PE(P) uses all 2n - 1 PM2I functions to route data. At each successive pass, all PEs that have data use all the PM2I interconnection functions to send the data. Since many copies of the same datum are passing through the network, collisions of data are not considered. In $|n/2|$ passes, the data has reached PE(Q), $0 \le Q < N$. []

VIII.6. Full Access Networks

Feierbach and Stevenson [FS77] introduced a Programmable Switching Network (PSN) which is topologically equivalent to a rearrangeable switching network [BE65] of size $N = 2^n$. Figure VIII.8 shows a PSN for $N = 8$ with the attendant controls. The $2n - 1$ stage network is built from four function interchange boxes. The first n stages are connected by Cube_0, Cube_1, ..., Cube_{n-1}. The next n-1 stages are the mirror image of the first n-1, and so are connected by Cube_{n-2}, Cube_{n-3}, ..., Cube_0.

Theorem VIII.15: The emulator network can simulate a 2n-1 stage PSN in 2n-1 stages.

Proof: Let c be the control for the interchange box associated with PE(i). An algorithm which simulates the 2n-1 Cube functions of a PSN is

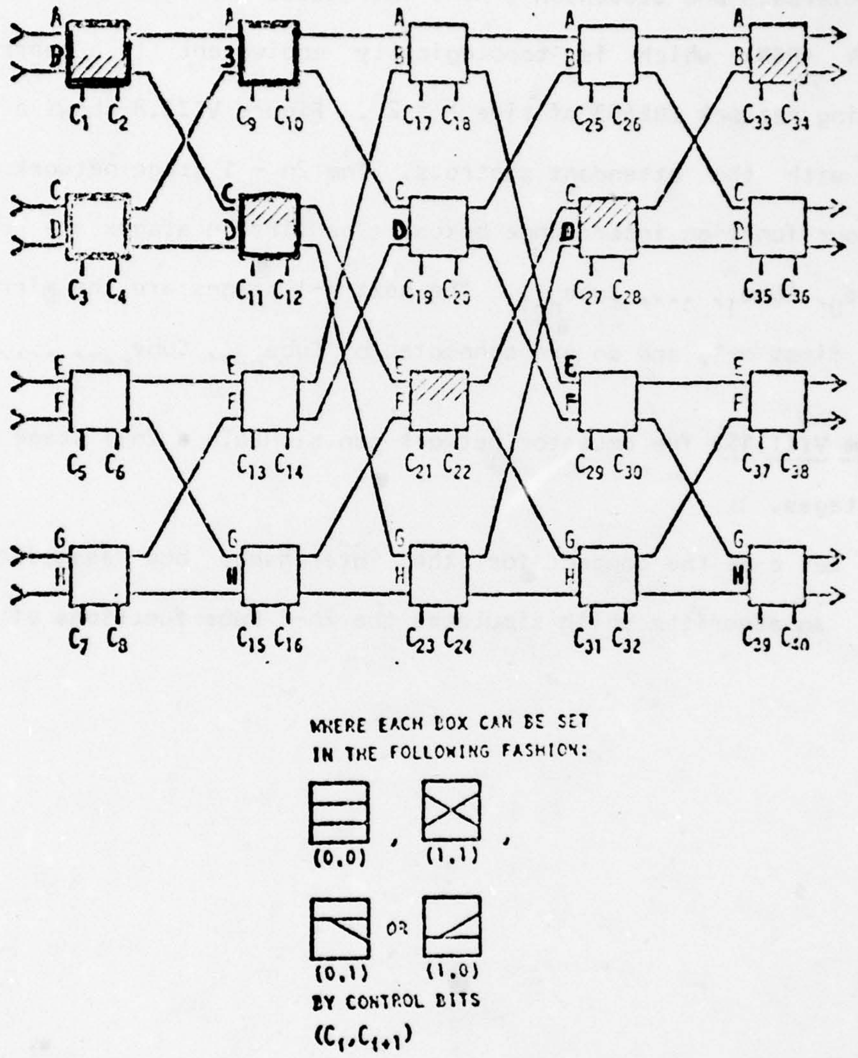


Figure VIII.8: A PSN for N = 8, and the control bits for each interchange box. [FS77]

```
activate all PEs;
RCR := 0;
for i = 0 step +1 until n-1 do
    if $p_i$ = 0 and c = ("exchange" or "upper broadcast")
            then RCR := $PM2_{+i}$;
    if $p_i$ = 1 and c = ("exchange" or "lower broadcast")
            then RCR := $PM2_{-i}$;
    TRANSFER;
    DTRin := DTRout;
for i = n-2 step -1 until 0 do
    if $p_i$ = 0 and c = ("exchange" or "upper broadcast")
            then RCR := $PM2_{+i}$;
    if $p_i$ = 1 and c = ("exchange" or "lower broadcast")
            then RCR := $PM2_{-i}$;
    TRANSFER;
    DTRin := DTRout.
```
□

VIII.7. Partitioning

In Chapter V, two types of partitioning were defined. It was shown how the recirculating PM2I network can be partitioned under single or multiple control partitioning. The PM2I functions of the emulator network partition in the same manner. For any recirculating or multistage network simulated above that can be partitioned, the emulator network can simulate that partitioning.

As an example, consider partitioning an indirect binary n-cube network for N = 8 into two groups of four PEs. Let group A be the PEs whose high-order address bits are 0, i.e., PEs 0, 1, 2, and 3. Let group B be the PEs whose high-order address bits are 1, i.e., PEs 4, 5, 6, and 7. Figure V.6 illustrates the needed controls for this partitioning. To simulate this partitioning, the emulator network must simulate the network shown in Figure VIII.9. An algorithm which simulates this partitioning is

```
activate all PEs;
RCR := 0;
for i = 0 step +1 until n-2
    calculate interchange box control;
    if (CONTROL = "Exchange") then $Cube_i$;
                            DTRin := DTRout.
```

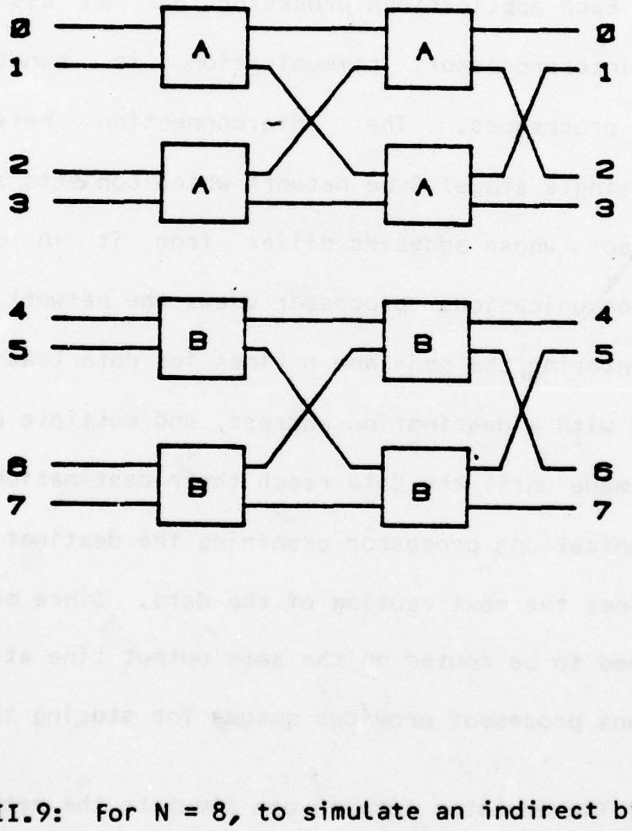


Figure VIII.9: For N = 8, to simulate an indirect binary n-cube that is partitioned into two groups based on the most significant PE address bits, the emulator system simulates the two stage network above.

VIII.8. The Emulator Network and MIMD Machines

Two recently proposed MIMD (multiple instruction stream - multiple data stream) systems are CHoPP and X-tree. In this section, the use of the emulator network to simulate the connection schemes of these systems is discussed.

The CHoPP (Columbia Homogeneous Parallel Processor) [SUB77] is a proposed MIMD system. Figure VIII.10 shows the physical configuration of this system. Each applications processor has an associated local memory. All interprocessor communication is handled by the communications processors. The interconnection network is a recirculating (single stage) Cube network which connects a processor to all other processors whose addesses differ from it in only one bit position. A communications processor views the network as $n = \log_2 N$ lines for data entering the node and n lines for data leaving the node. Data are passed with a destination address, and multiple passes through the network are made until the data reach their destinations. At each pass, the communications processor examining the destination address of the data determines the next routing of the data. Since more than one data item may need to be routed on the same output line at any one time, the communications processor provides queues for storing these data.

Theorem VIII.16: The emulator network can simulate the method of routing data used in CHoPP.

Proof: Partition the emulator system into two groups of PEs, the applications PEs and the communications PEs. Let the applications PEs be those whose least significant address bit is 0, and let the

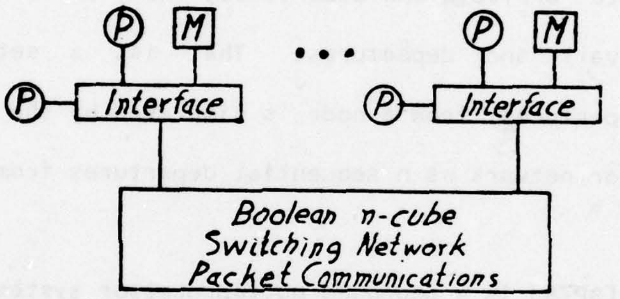


Figure VIII.10: The CHoPP is a proposed MIMD computer system. The interconnection network routes data between processing elements [SUB77].

communications PEs be those whose least significant address bit is 1. Data is routed between applications PEs and communications PEs by Cube_0. The communication PEs are connected using the recirculating Cube network functions, which the emulator network can perform (Theorem VIII.9).

The communications PEs have the ability to simulate queues for storing data and to examine the destination tags to determine the next routing. Since only one DTRin and one DTRout register are available to each PE, the emulator network cannot route more than one datum from a communications PE or accept more than one datum in one CHoPP time period. Multiple arrivals and departures, then, are simulated as sets of single arrivals and departures. That is, a set of, say, n simultaneous departures from a node is simulated by the communications PE of the emulator network as n sequential departures from the node. []

The X-tree [DP78] is a proposed multiprocessor system in which the processors are interconnected in a tightly coupled, hierarchical, binary tree structured network. Redundant links are added to the binary tree of PEs to provide potential fault tolerant communications. Two types of added links are discussed in [SDP78]. Figure VIII.11 shows a half-ring structure where the links added to the binary tree at level i are between PE(P) to PE(P'), where for P even, P' is the PE to the "left" of P on level i. Figure VIII.12 shows a full-ring topology, where the added links connect all PEs to the PE on the "right" as well as the PE on the "left". In the full-ring topology, for a 5-level tree with $2^5 - 1 = 31$ PEs, the PE addresses and links for the emulator network simulation are shown in Figure VIII.13. The PE at level 0, the root

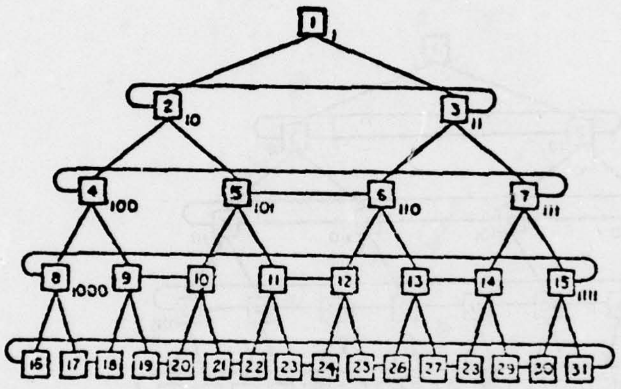


Figure VIII.11: The half-ring tree structure has been suggested for connecting processors in the X-tree system [SDP78].

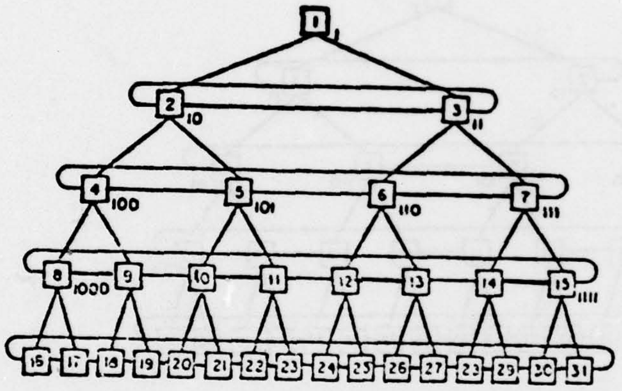


Figure VIII.12: The full-ring tree structure has been proposed for connecting processors in the X-tree system [SDP78].

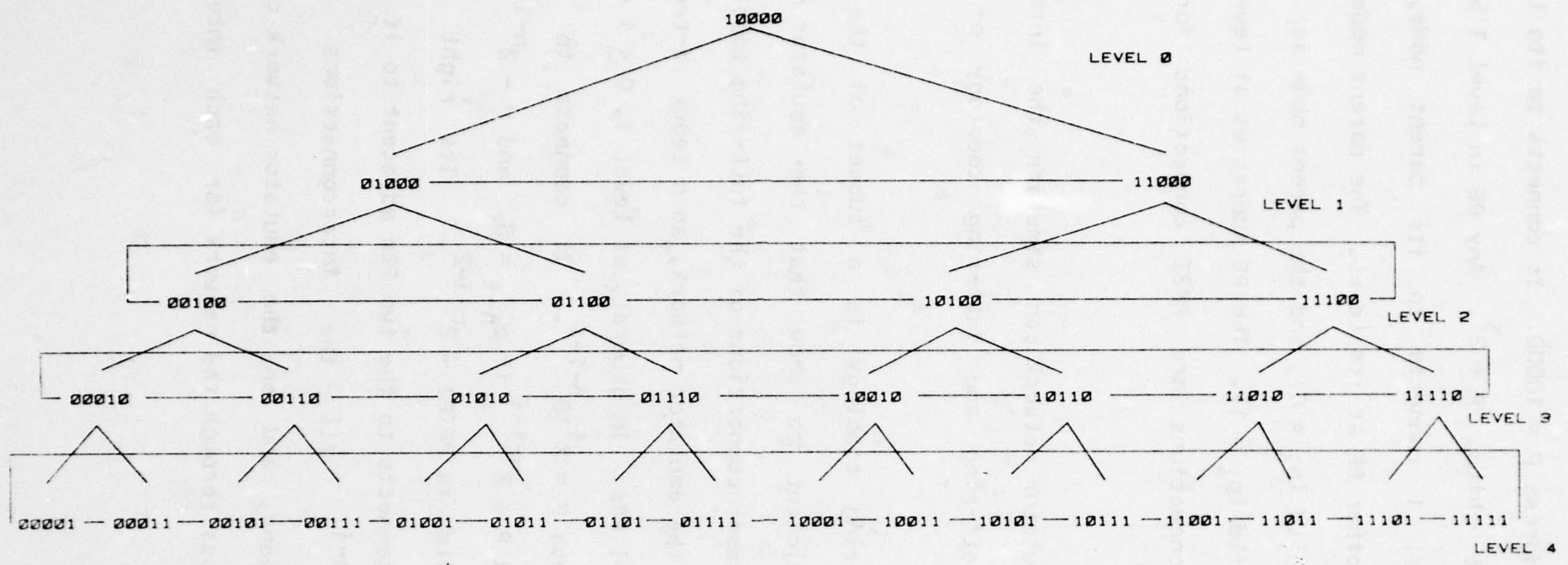


Figure VIII.13: N = 31 PEs can be connected in a full-ring structure where the addresses at each level are as given.

node of the tree, has address P = 10000. It connects to its left child, $P - 2^3$, and its right child, $P + 2^3$. Any PE in level 1 has address P = X1000. A PE in level 1 connects to its parent node, its two children, and to the other PE at its level. The parent node is $P + 2^3$ if the PE is the left child ($p_4 = 0$), and the parent node is $P - 2^3$ if the PE is the right child ($p_4 = 1$). The PE addresses at level 1 differ by 2^4. Note that all connections are PM2I connections for the PEs selected for the tree.

<u>Theorem</u> <u>VIII.17</u>: The emulator network can simulate the interprocessor connections of the half-ring and full-ring topology of the X-tree multiprocessor system.

<u>Proof</u>: Since the half-ring topology is a subset of the full-ring topology, it is sufficient to show that the emulator network can simulate the interprocessor connections of the full-ring topology.

For $N = 2^n$ PEs of the emulator network, an n level X-tree can be constructed that has N-1 PEs. In general, at level i, $0 \leq i < n$, any PE at that level has address $P = X^i 10^{n-1-i}$. It connects to five other PEs. Its parent PE is $P + 2^{n-1-i}$ if $p_{n-i} = 0$, and $P - 2^{n-1-i}$ if $p_{n-i} = 1$. Its left child is $PE(P) - 2^{n-i-2}$. Its right child is $PE(P) + 2^{n-i-2}$. It connects to the two PEs adjacent to it at level i, $P + 2^{n-i}$ and $P - 2^{n-i}$. All the interconnections are PM2I interconnection functions, and so the emulator network can simulate their effects in one pass through the network for each interconnection function. □

VIII.9. Conclusions

The emulator network, with N inputs and outputs, can be built from $N(2 \log_2 N + 1)$ tri-state buffers, for each bit of network width. It can simulate the following $n = \log_2 N$ stage networks in n passes: Pease's indirect binary n-cube, Goke and Lipovski's n-stage SW banyon and CC banyon, Batcher's STARAN flip network with flip or shift control, Siegel and Smith's generalized cube, n-stage Shuffle-Exchange, Lang and Stone's multistage Shuffle-Exchange with "simplified control," Lawrie's omega with destination tag control, Lawrie's omega with broadcast capabilities, Feng's data manipulator, and Siegel and Smith's augmented data manipulator. In one pass, the emulator can simulate an interconnection function of the following recirculating (single stage) networks: Shuffle-Exchange, Cube, Plus-Minus 2^i (PM2I), and Illiac. It can simulate the n-stage Shuffle - No Shuffle - Exchange (SNSE) network in at most 2n passes and Feierbach and Stevenson's 2n-1 stage Programmable Switching Network (PSN) in at most 2n-1 passes. It can simulate a shuffle or an inverse shuffle of size 2^{n-k} in n-k-1 passes. The emulator network can also simulate the partitioning of any of the above networks that have been shown to be partitionable.

The flexibility of the emulator network is seen in its ability to simulate many different control schemes for many types of networks. This flexibility comes from the types of interconnection functions included and from the independent function control that allows each PE to execute any of the 2n interconnection functions of the network independently of any other PEs. Given the logic functions at each node of a test network to be emulated, the PEs at each node of the emulator network can calculate these logic functions and control the emulator network so that it responds in the same manner as the test network. The emulator network can also be used to test for the effects of faulty nodes in the network. A given node of the emulator network can be assigned to fail in a predetermined manner. The effects of the failure on the remainder of the network can then be observed. Because of this flexibility, the emulator network can be a powerful tool for evaluating proposed SIMD machine interconnection networks and their control schemes.

IX. CONCLUSIONS

This work is a study of interconnection networks for SIMD machines, with special emphasis placed on reconfigurable, partitionable, multiple SIMD systems. The results have both immediate practical applications and theoretical implications about the nature of multiprocessor interconnection systems.

The partitioning of interconnection networks (Chapter V) is essential knowledge for designing partitionable multiple SIMD computer systems. In exploring this topic, information was gained about the types of data transfers that networks can perform and the limits imposed by the interconnection functions of the networks and the type of control scheme used. For example, for $N = 2^n$, the indirect binary n-cube network with individual interchange box control can be partitioned into 2^{n-r} smaller, complete, independent indirect binary n-cube networks when all PE address bits within a partition have n-r address bits in common. The STARAN flip network is topologically equivalent to the indirect binary n-cube [SIS78]. The flip control allows the flip network to partitioned only under single control partitioning, where all partitions must perform the same data transfer. With the more flexible shift control, the flip network cannot be partitioned under multiple control partitioning, where all partitions may perform different data transfers.

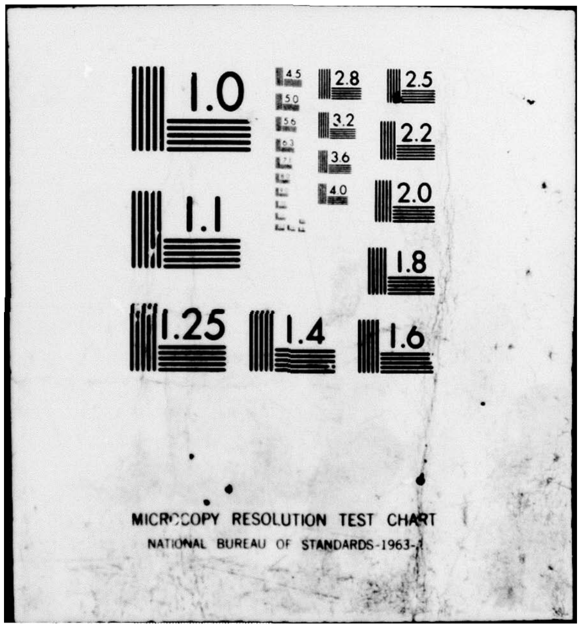
Under single control partitioning, where all partitions perform the same data transfer, the shift control can only partition the flip network into 2^{n-r} groups if n-r of the most significant or the least significant address bits are the same for all PEs of the partition. Consider also the generalized cube network with individual interchange box control. The generalized cube can be partitioned under the restriction that n-r PE address bits are the same for all PEs in the partition. The Augmented Data Manipulator, which is a more powerful interconnection network than the generalized cube (Chapter VI) can be partitioned only when some set of the least significant address bits are common to all PEs in a partition. Future work in this area should consider the entire computer system and the influence the interconnection network exerts upon it. A partitionable system has more overhead in operating system software than a conventional SIMD machine. Also, extra hardware costs will be incurred to control the partitioned network.

The emulator network is presented as a practical hardware design aid. The theorems of Chapter VIII are also theoretical results about the nature of the interconnection networks that were discussed. The PM2I functions, together with independent function control and the computation facilities of the PEs, can be used to simulate many interconnection networks from the literature, as well as useful interconnections such as the perfect shuffle, the inverse perfect shuffle, and broadcasts. When designing a prototype for a partitionable SIMD parallel processing system, incorporation of the emulator network into the prototype would allow different network connections and control schemes to be tested and evaluated. The algorithms that use the

AD-A078 690 PURDUE UNIV LAFAYETTE IN SCHOOL OF ELECTRICAL ENGINEERING F/G 9/2
DESIGN AND ANALYSIS OF INTERCONNECTION NETWORKS FOR PARTITIONAB--ETC(U)
AUG 79 S D SMITH , H J SIEGEL AFOSR-78-3581
UNCLASSIFIED TR-EE79-39 AFOSR-TR-79-1272 NL

4 OF 4
AD
A078690

END
DATE
FILMED
1 — 80
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

emulator network to simulate other networks reveal information about the characteristics of the networks being simulated. Thus, this chapter provides both a practical and theoretical analysis.

The study of the Augmented Data Manipulator in Chapter VI enumerates some of the differences between the ADM and multistage cube networks. Theorems are presented which show how and why the ADM can perform a perfect shuffle function in one pass through the network, but not an inverse perfect shuffle or a bit reversal in a single pass. Some fault tolerant properties of the network are presented. The Cube functions and the uniform shift functions are used to show how a given interconnection may have several different, equivalent paths through the ADM network. The relationship between the ADM and Inverse ADM is also analysed.

Chapter IV developed techniques for the analysis of combinational logic multistage networks, pipelined multistage networks, and recirculating networks. Detailed comparisons were made between combinational logic multistage and pipelined multistage networks for networks of equal cost and networks that require equal time to complete a given data transfer. Examples were presented which illustrate the behavior of these networks for a given cost and for a given number of data items to be transferred. These analyses will aid a system designer in choosing a network implementation which will be cost-effective for the intended application of the system.

These results were applied in Chapter VII to three parallel algorithms for image processing. For each algorithm, the impact on the computation time of different choices of interconnection networks was shown. When uniform shifts of the data from one PE to another are present in the algorithm, the recirculating cube is a poor choice of interconnection network. A binary addition algorithm must be used to calculate each data transfer. Multistage cube and PM2I networks, however, can perform such uniform shifts in one use of the network. The Illiac network is useful for passing data to PEs in the immediate neighborhood of PE P, i.e., PEs (P+1) modulo N, (P-1) modulo N, (P+-/N) modulo N, and (P--/N) modulo N. When data transfers are among PEs outside a limited neighborhood, as in the histogram formation algorithm presented, the Illiac network performs poorly. When transfers of blocks of data are common in an algorithm, a pipelined multistage network shows its worth. Future work in this area should examine more algorithms for the types of data transfers present. From this, models of data transfers can be developed, and techniques for evaluation of the impact of the interconnection network on the algorithm can be applied.

Tools were developed which can be used by the architect of parallel computer systems to design intelligently an interconnection network. Underlying theoretical aspects of interconnection networks presented here further expand the knowledge of the types of interprocessor connections that networks can accomplish. The analysis of the structure of networks has resulted in criteria which can be applied to determine the usefulness of a given network for a specific task.

LIST OF REFERENCES

[BAR68] Barnes, G. H., Brown, R. M., Kato, M., Kuck, D. J., Slotnick, D. L., and Stokes, R. A., "The Illiac IV computer," IEEE Transactions on Computers, vol. c-17, no. 8, August 1968, pages 746-757.

[BA68] Batcher, K. E., "Sorting networks and their applications," Proceedings of the Spring Joint Computer Conference 1968, AFIPS Conference Proceedings, volume 32 (1968), pages 307-314.

[BA75] Batcher, K. E., "The multidimensional access memory in STARAN," IEEE Transactions on Computers, vol. c-26, no. 2, February 1977, pages 174-177.

[BA76] Batcher, K.E. , "The flip network in STARAN," Proceedings of the 1976 International Conference on Parallel Processing, August 1976, pages 65-71.

[BAU74] Bauer, L. H., "Implementation of data manipulating functions on the STARAN associative array processor," Lecture Notes in Computer Science 24, Proceedings of the 1974 Sagamore Computer Conference on Parallel Processing, T. Feng, editor, Springer-Verlag, Berlin, August 1974, pages 209-227.

[BE65] Benes, V. E., Mathematical Theory of Connecting Networks and Telephone Traffic, New York: Academic, 1965.

[BFHP79] Briggs, F., Fu, K. S., Hwang, K., and Patel, J., "PM4: A reconfigurable multiprocessor system for pattern recognition and image processing," 1979 National Computer Conference, AFIPS Conference Proceedings, volume 48, June 1979, pages 255-265.

[BOU72] Bouknight, W. J., Denenberg, S. A., McIntrye, D. E., Randall, J. M., Sameh, A. H., and Slotnick, D. L., "The Illiac IV System," Proceedings of the IEEE, vol. 60, no. 4, April 1972, pages 369-388.

[BS78] Bogdanowicz, J. F., and Siegel, H. J., "A partitionable multi-microprogrammable-microprocessor system for image processing," Proceedings of the IEEE Computer Society Workshop on Pattern Recognition and Artificial Intelligence, April 1978, pages 141-144.

[CDL77] Cordella, L., Duff, M. J. B., and Levialdi, S., "Thresholding: a challenge for parallel processing," Computer Graphics and Image Processing, vol. 6, 1977, pages 207-220.

[CDL78] Cordella, L. P., Duff, M. J. B., and Levialdi, S., "An analysis of computational cost in image processing: a case study," IEEE Transactions on Computers, vol. c-27, no. 10, October 1978, pages 904-910.

[CGY74] Couranz, G. R., M. S. Gerhardt, and C. J. Young, "Programmable RADAR signal processing using the RAP," Lecture Notes in Computer Science 24, Proceedings of the 1974 Sagamore Computer Conference on Parallel Processing, T. Feng, editor, Springer-Verlag, Berlin, August 1974, pages 37-52.

[DP78] Despain, A. M. and Patterson, D. A., "X-tree: a tree structured multi-processor computer architecture," Fifth Annual Symposium on Computer Architecture (ACM SIGARCH Newsletter, vol. 6, no. 7, April 1979), April 1978, pages 144-151.

[FE74] Feng, T. , "Data manipulating functions in parallel processors and their implementations," IEEE Transactions on Computers, vol. C-23, no. 3, March 1974, pages 309-318.

[FL66] Flynn, M. J., "Very high-speed computing systems," Proceedings of the IEEE, vol. 54, no. 12, December 1966, pages 1901-1909.

[FS77] Feierbach, G., and Stevenson, D., "A feasibility study of programmable switching networks for data routing," Phoenix Project, memorandum number 003, Institute for Advanced Computation, Sunnyvale, California.

[FU78] Fu, K. S., "Special computer architectures for pattern recognition and image processing - an overview," 1978 National Computer Conference, AFIPS Conference Proceedings, volume 47 (1978), pages 1003-1013.

[GEN78] Gentleman, W. M., "Some complexity results for matrix computations on parallel processors," Journal of the ACM, vol. 25, no. 1, January 1978, pages 112-115.

[GL73] Goke, L. R., and Lipovski, G. J., "Banyon networks for partitioning multiprocessor systems," First Annual Symposium on Computer Architecture (ACM SIGARCH Newsletter, vol. 2, no. 4, December 1973), December 1973, pages 21-28.

[GOK76] Goke, L. Rodney, "Banyon networks for partitioning multiprocessor systems," PhD thesis, Department of Electrical Engineering, University of Florida, June 1976.

[GOL61] Golumb, S. W., "Permutations by cutting and shuffling," SIAM Review, vol 3, October 1961, pages 293-297.

[GR76] Graham, Marvin Lowell, "An array computer for the class of problems typified by the general circulation model of the atmosphere," PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, January 1976.

[HEL77] Heller, D., "Minimal parallelism for computations under time constraints," Symposium on High Speed Computer and Algorithm Organization, D. J. Kuck, D. H. Lawrie, A. H. Sameh, editors, Academic Press, New York, April 1977, pages 321-322.

[HIG72] Higbie, L. C., "The Omen computer associative array processor," Compcon 72, IEEE Computer Society Conference Proceedings, September 1972, pages 287-290.

[KST73] Kogge, P. M., and Stone, H. S., "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Transactions on Computers, vol. c-22, no. 8, August 1973, pages 786-793.

[KUC77] Kuck, D. J., "A survey of parallel machine organization and programming," ACM Computing Surveys, vol. 9, March 1977, pages 29-59.

[LAN76] Lang, T., "Interconnections between processors and memory modules using the shuffle-exchange network," IEEE Transactions on Computers, vol. c-26, no. 5, May 1976, pages 496-503.

[LAST76] Lang, T., and Stone, H. S., "A shuffle-exchange network with simplified control," IEEE Transactions on Computers, vol. c-25, no. 1, January 1976, pages 55-65.

[LAW73] Lawrie, Duncan H., "Memory - processor connection networks," PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, February 1973.

[LAW75] Lawrie, D. H., "Access and alignment of data in an array processor," IEEE Transactions on Computers, vol. c-24, no. 12, December 1975, pages 1145-1155.

[LT77] Lipovski, G. T., and Tripathi, A., "A reconfigurable varistructure array processor," 1977 International Conference on Parallel Processing, August 1977, pages 165-174.

[NU77] Nutt, G. J., "Microprocessor implementation of a parallel processor," Fourth Annual Symposium on Computer Architecture (ACM SIGARCH Newsletter, vol. 5, no. 7, March 1977), March 1977, pages 147-152.

[OPT71] Opferman, D. C., and N. T. Tsao-Wu, "On a class of rearrangeable switching networks," Bell System Technical Journal, vol. 50, May-June 1971, pages 1579-1618.

[PAT79] Patel, J. H., "Processor-memory interconnections for multiprocessors," Sixth Annual Symposium on Computer Architecture (ACM SIGARCH Newsletter, vol. 7, no. 6, April 1979), April 1979, pages 168-177.

[PEA77] Pease, M. C. , "The indirect binary n-cube microprocessor array," IEEE Transactions on Computers, vol. c-26, no. 5, May 1977, pages 458-473.

[REI60] Reitwiesner, G. W., "Binary Arithmetic," in Advances in Computers, volume 1, Academic Press, New York, New York, 1960.

[ROK76] Rosenfeld, A., and Kak, A. C., Digital Picture Processing, Acedemic Press, New York, 1976.

[SAM77] Sameh, A. H., "Numerical parallel algorithms - A survey," Symposium on High Speed Computer and Algorithm Organization, D. J. Kuck, D. H. Lawrie, A. H. Sameh, editors, Academic Press, New York, April 1977, pages 207-226.

[SDP78] Sequin, C. H., Despain, A. M., and Patterson, D. A., "Communication in X-tree, a modular multiprocessor system," ACM 78 Proceedings, December 1978, pages 194-203.

[SIE75] Siegel, H. J., "Analysis Techniques for SIMD machine interconnection networks and the effects of processor address masks," 1975 Sagamore Computer Conference on Parallel Processing, August 1975, pages 106-109.

[SIE77a] Siegel, H. J. , "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," IEEE Transactions on Computers, vol. c-26, no. 2, February 1977, pages 153-161.

[SIE77b] Siegel, H. J., "Interconnection networks and masking schemes for single instruction stream - multiple data stream machines," PhD thesis, Department of Electrical Engineering and Computer Science, Princeton University, May 1977.

[SIE78a] Siegel, H. J., "Preliminary design of a versatile parallel image processing system," Proceedings of the Third Biennial Conference on Computing in Indiana, April 1978, pages 11-25.

[SIE78b] Siegel, H. J., "Partitionable SIMD computer system interconnection network universality," Proceedings of the Sixteenth Annual Allerton Conference on Communication, Control, and Computing, October 1978, pages 586-595.

[SIE79a] Siegel, H. J., "Interconnection networks for SIMD machines," IEEE Computer Magazine, vol. 12, no. 6, June 1979, pages 57-65.

[SIE79b] Siegel, H. J., "A model of SIMD machines and a comparison of various interconnection networks," to appear in IEEE Transactions on Computers.

[SIS78] Siegel, H.J., and Smith, S. D., "Study of multistage SIMD interconnection networks," Fifth Annual Symposium on Computer Architecture (ACM SIGARCH Newsletter, vol. 6, no. 7, April 1979), April 8-10, 1978, pages 223-229.

[SM78] Siegel, H. J., and Mueller, Jr., P. T.,, "The organization and language design of microprocessors for an SIMD/MIMD system," Second Rocky Mountain Symposium on Microcomputers: Systems, Software, Architecture, August 1978, Pingree Park, Colorado, pages 311-340.

[SMSM78] Siegel, H. J., Mueller, Jr., P. T., Smalley, Jr., H. E., "Control of a partitionable multimicroprocessor system," 1978 International Conference on Parallel Processing, August 1978, pages 9-17.

[SS79] Siegel, H. J., Siegel, L. J., McMillen, R. J., Mueller, Jr., P. T., and Smith, S. D., "An SIMD/MIMD multimicroprocessor system for image processing and pattern recognition," 1979 Conference on Pattern Recognition and Image Processing, August 1979.

[ST71] Stone, H. S., "Parallel processing and the perfect shuffle," IEEE Transactions on Computers, vol. c-20, no. 2, February 1971, pages 153-161.

[ST75] Stone, H. S., "Parallel Computers," in Introduction to Computer Architecture, Science Research Associates, Inc., Chicago, Illinois, 1975, pages 318-374.

[SUB77] Sullivan, H., and Bashkow, T. R., "A large scale, homogeneous, fully distributed parallel machine, I," Fourth Annual Symposium on Computer Architecture (ACM SIGARCH Newsletter, vol. 5, no. 7, March 1977), March 1977, pages 105-117.

[SW78] Swain, P. H., "Fundamentals of Pattern Recognition in Remote Sensing," Chapter 3 in P. H. Swain and S. M. Davis, editors, Remote Sensing: The Quantitative Approach, McGraw-Hill, 1978.

[TI76] The TTL Data Book for Design Engineers, Texas Instruments, Dallas, Texas, 1976.

[WEN76] Wen, K. Y., "Interprocessor connections -- capabilities, exploitation, and effectiveness," Department of Computer Science, University of Illinois, Urbana, Illinois, Report no. UIUCDCS-R-76-830, October 1976.

[WES72] Wester, A. H., "Special features in SIMDA," Proceeding of the 1972 Sagamore Computer Conference, Rome Air Development Center and Syracuse University, August 1972, pages 29-40.

Errata

1. Figure IV.18 caption belongs to the figure with Figure IV.20 caption.

2. Figure IV.19 caption belongs to figure with IV.18 caption.

3. Figure IV.20 caption belongs to figure with IV.21 caption.

4. Figure IV.21 caption belongs to figure with IV.19 caption.

5. page 151, line 15 should be: then $PM_{+2}(P) = Q$ and $PM_{-2}\,(Q) = P$.